The Wireless Communications Transfer Protocol (WCTP)

Protocol Specification

Revision: Version 1.3 **Update 1**

Release Date: Version 1.3 April, 22 2005
Update 1 May 23 2006

NOTICE

Paging Technical Committee (PTC) Engineering Standards and Publications are designed to serve the public interest through eliminating misunderstandings between manufacturers and purchasers, facilitating interchangeability and improvement of products, and assisting the purchaser in selecting and obtaining with minimum delay the proper product for their particular need. Existence of such Standards and Publications shall not preclude in any respect any member or non-member of the PTC from manufacturing or selling products not conforming to such Standards and Publications, nor shall the existence of such Standards and Publications preclude their voluntary use by those other than PTC members, whether the standard is to be used either domestically or internationally.

Standards and Publications are adopted by the PTC without regard to whether or not their adoption may involve patents or articles, materials, or processes. By such action, the PTC does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the Standard or Publication.

PTC INTERIM STANDARDS

PTC Interim Standards contain information deemed to be of technical value to the industry, and are published at the request of the originating Committee without necessarily following a rigorous public review and resolution of comments.

This Interim Standard was prepared by the PTC, Device Standards Group, and draws upon the ReFLEX™ Standard itself.

Published by
Paging Technical Committee

Trademarks

Table of Contents

List of Tables

List of Figures

1 Introduction

The Wireless Communications Transfer Protocol (WCTP) was designed to simplify the transmission of alphanumeric and binary messages from wire line systems and two-way devices. WCTP is equally effective and accessible with bi-directional messaging devices. The robust nature of WCTP brings enhanced capabilities to wireless technologies especially in the Narrowband PCS sector.

The Messaging Standards Committee, an ad hoc committee, submitted a draft proposal to the Radio Paging Community, to foster industry input, collaboration, technical study, industry promotion, and participation in the further expansion of an open non-proprietary standard. The Messaging Standards Committee is a group of paging industry manufacturers and carriers focused on rapidly creating mutually agreeable standards to address emerging applications, which cannot be easily realized through the utilization of existing standards. When the first proposal was received, a sub-committee was established to improve the protocol and to issue it as a specification. The sub-committee has been moved into the Paging Technical Committee (PTC). The effort continues to refine the capabilities and features of this protocol. Furthermore, as development continues additional functionality benefits the industry and provides remarkable capabilities. The drafting committee's intent is to maximum the scope of future revisions while maintaining compatibility with existing protocols. The expected benefit of seamless incorporation of new features will be coupled with a means of facilitating communication with different protocols. The Personal Communications Industry accepted the first full release and has adopted the protocol as a PCIA standard. On-going development of WCTP continues within the PTC's Protocol Working Group (PWG).

The Wireless Industry hosts a wide range of diverse protocols that vary in complexity and capability to submit messages within a wireless network (e.g. TAP, TNPP, TDP/TME, TME-X, WMtp™, WMapi™, UCP, I4, and SMPP). When messages are submitted within the structure of Internet Standard protocols such as the Simple Mail Transport Protocol (SMTP/Email protocol) or the Hypertext Transfer Protocol (HTTP), the information conveyed to the wireless network is limited. The exchange limitation is a derivative often caused by generalized protocols. The problem has been compounded when carriers create proprietary software that is network specific. Moreover, the implementation of "wireless enabled" applications creates the ~~urge~~ need to find the correct protocol for integration within their network applications. Presently, easy portability across wireless networks, applications, and protocols does not exist.

WCTP provides the wireless industry specific standardization within messaging:

- When messages are being sent from a wire line host to a wireless device.
- When messages are being sent from a wireless device to a wire line host.
- Facilitates inter-carrier connectivity for messages sent from a wireless device to a wireless device on another carrier's network.

1.1 Goals of WCTP

The major goals to be derived through WCTP are:

- To interconnect Internet accessible systems with wireless networks
- To utilize current and emerging Internet standards that will dramatically simplify the process of implementing WCTP
- To send and receive messages containing any type of binary data content
- To support as many wireless networks as possible
- To create an extensible architecture allowing unlimited growth to accommodate new features and capabilities
- To support a wide range of wireless devices
- To support the ability for wireless devices to send messages to other wireless devices located on other wireless carriers' networks

1.2 Future of WCTP

Many suggestions have been made to improve WCTP. It is not known which will be implemented in future versions. The primary goals are to simplify use and practical application of WCTP.

These suggestions include:

- Splitting the protocol into a suite with logical functional groupings
- Organizing a single repository for aliasing long addresses in order to save on wireless bandwidth
- Adding device or wireless protocol specific instructions for addressing
- Implementing WCTP to be validated against an XML schema rather than a DTD

1.3 WCTP and its relationship to WAP

The Wireless Application Protocol forum has received worldwide attention and at the time of this writing has grown to a membership of more than 85 corporations. A set of more than 50 specification documents currently exists detailing the architecture and protocol specifics of the various aspects of the "WAP Protocol Suite". WAP is still an evolving specification. Many people, who are not knowledgeable about all of the current WAP specifics, believe that WAP is the all-encompassing answer to two-way communications with wireless devices. In reality, there are still networks and wireless applications that have been designed and implemented without WAP, and would still need additional protocols to be defined in order to utilize the maximum benefits of those networks or applications.

The initial focus of WAP has concentrated on the support of a handheld wireless Internet browser and integrated telephony/browser functions as well as a standard set of protocols between the handset, the wireless network and the WAP Gateway to support these functions. The set of protocols established by the WAP forum are designed to support much more than browser functions. WAP discusses the desire to access the various layers of the protocol suite through a set of standardized API's, but these have not yet been fully established.

WCTP is complementary to WAP in that it addresses a standardized manner to move data to/from the wireless network gateway over a wired network (an area noted as not being addressed). It might even represent a means of presenting data directly to the internal WAP layers. As such, WCTP could potentially be studied for integration with the WAP protocol stack. Having such discussions is anticipated.

2 Protocol Prerequisites

WCTP was developed to provide optimum functionality for wireless messaging by leveraging well-known standards. You should be familiar with all of these other standards before attempting to implement this protocol.

The WCTP protocol defines a number of "operations," each of which is represented as a “control block” in an XML form. The operations currently supported by the protocol are defined throughout this specification. The names of the various operations are of the form "`wctp-OperationName`".

2.1 References

The following documents and Web site addresses provide information that is pertinent to this specification.

- Reference information regarding XML, Web site: http://www.w3.org/XML
- RFC 822, Standard For the Format Of ARPA Internet Text Messages: http://www.ietf.org/rfc/rfc0822.txt
- RFC 1341, MIME (Multipurpose Internet Mail Extensions): http://www.ietf.org/rfc/rfc1341.txt
- RFC 1738, Uniform Resource Locators (URL): http://www.ietf.org/rfc/rfc1738.txt
- RFC 1945, Hypertext Transfer Protocol -- HTTP/1.0: http://www.ietf.org/rfc/rfc1945.txt
- RFC 2068, Hypertext Transfer Protocol -- HTTP/1.1: http://www.ietf.org/rfc/rfc2068.txt
- RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax: http://www.ietf.org/rfc/rfc2396.txt
- Motorola, FLEXsuite™ of Enabling Protocols, Document Number: 6881139B10
- Motorola, ReFLEX™ 25 Specification, Document Number: RFL25LX-033195
- Personal Communications Industry Association Web site: http://www.pcia.com
- WAP Forum Web Site: http://www.wapforum.org
- ISO8601, Web Site: http://www.iso.org/
- WCTP Working Group Web Site: http://www.wctp.org

2.2 WCTP Communications through HTTP

WCTP is called a “transfer” protocol in that it is transferring information content between wire line and wireless systems. The manner in which the WCTP-defined operations are transported between a pair of systems is independent of WCTP syntax.

Although any number of transport protocols may be used to move WCTP operations between systems, the Hypertext Transfer Protocol (HTTP) has been selected as the recommended transport protocol for WCTP. HTTP was selected because its use already addresses the issues of security and firewall protection typically found within those corporations that allow Internet access and that are expected to be the primary users of the protocol. The firewalls maintained by the security departments of most corporations normally allow HTTP requests to be initiated from systems that are within a corporate network. Because of this, WCTP will be able to be deployed in most corporate environments without significant coordination with the security administration of the network.

2.3 WCTP Through Other Transport Mechanisms

HTTP does not address issues of privacy and security in the transport of WCTP messages. Other secure transport mechanisms can be used with WCTP. The availability of other transport mechanisms may vary between carriers.

2.3.1 Secure Communication through HTTPS

HTTPS can be used to provide secure communication between a client and server over the public internet. This would be implemented at the WCTP server to secure the internet link between the client and the server. Technically HTTPS is normal HTTP running over an encrypted secure socket layer (SSL) transport mechanism.

The main benefits of using HTTPS are:

- The plain text of the WCTP message is not transmitted over the internet as it is with HTTP. This ensures the privacy of communications as intercepted messages will contain only the encrypted message.
- If a public certificate (for example VeriSign, SecureSign, etc.) is used, the client will be able to verify the identity of the WCTP server to which it is sending messages.
- If the client implements HTTPS then messages sent from the carrier back to the client can also be secure. Polling clients would not need to do this.

2.3.2 Secure Communication through VPN

If greater security is required, then the WCTP connection could be implemented with a VPN (Virtual Private Network).

The main benefits of using VPN are:

- Both originator and recipient are mutually authenticated
- More secure than HTTPS: Carriers could create a virtual pipe between the devices and the client, preventing any one else from sending messages to the devices.
- VPN provides a virtual link from the client to the wctp gateway. Messages are encrypted, source and destination are also secure.

2.4 XML Syntax

WCTP uses XML as the format to define its operations. WCTP operations must be formatted in accordance with the XML 1.0 specification. Furthermore, these operations must be "well-formed" and “valid” according to the WCTP Data Type Definition. The meaning of well formed and valid is described in the XML specifications referenced. If you are not familiar with XML, please familiarize yourself with XML before continuing.

2.5 Status and Error Codes

WCTP standardizes all of the status and error codes returned in a WCTP Response operation. It utilizes the familiar Internet protocol paradigm of three-digit status values in responses to protocol operations. This paradigm was chosen because it is well understood and is suited to both machine-to-machine communication and human interpretation.

2.5.1 No HTTP Relationship

There is no relationship between the status codes in WCTP and the status codes at the HTTP transport level. As described above, HTTP is merely an instance of a transport mechanism for WCTP operations and any WCTP implementations utilizing HTTP must appropriately honor HTTP status or error conditions at the transport level. The semantics of completing the HTTP transport operation do not affect the semantics of the WCTP operations in any way.

Throughout the rest of this discussion, an HTTP status of "200 OK" (or "100 CONTINUE," if applicable) is implicit for the transport of the WCTP operations.

2.5.2 Your Responsibility

WCTP implementation will require an application, not just a web page on a remote server. To correctly implement the request and response behavior, you must accept and understand the response a WCTP server will return to you as a client. If you are building a server to accept WCTP transactions initiated by a remote host you must be able to accept a valid WCTP document as well as return one to the remote host in the same HTTP session.

2.5.3 Status and Error Code Returned Text Strings

WCTP Response operations that return success or error codes are restricted to specifying one of the standard error codes as listed in Appendix E Error and Success Codes. The error and success responses may optionally return an error text or success text attribute. The text string returned in this attribute is implementation-specific, but must be consistent with the general description of the associated status or error code value (specified in Appendix E Error and Success Codes) that is required in status response operations. WCTP implementations must process the three-digit numeric value returned. WCTP implementations should not use the text string returned for any purposes other than as informational data with no defined semantics. A typical use for the text string is to report it to an error or call processing log file for later analysis by humans. WCTP implementations must not utilize the free-form body of the WCTP status or error response for any other purpose than as informational data with no defined semantics. Again, a typical use for the text string is to report it to an error or call processing log file for later analysis by humans. For error responses, the free-form data returned may assist in tracking down XML data encoding problems in WCTP implementations.

Error and success responses may optionally return an arbitrary text string within the error or success element. In the case of error responses, this text string could contain portions of the WCTP XML syntax where errors were found resulting in the inability to process the requested operation. For example, the following XML fragment could be returned when a date value is determined to be incorrect:

```
<wctp-Failure errorCode="404"
              errorText="Invalid date/time format">
   deliveryBefore="January 29, 1999" is not a valid WCTP
   date/time format
</wctp-Failure>
```

2.5.4 Categories of Status and Error Codes

The defined status and error codes are specified in Appendix E Error and Success Codes of this specification. Each code contains a three-digit numeric value and a general description of the status or error code.

Implementations of WCTP may treat unrecognized codes not defined in Appendix E Error and Success Codes of the WCTP specification or any 9xx series code in an implementation-specific manner.

The general categories into which a status or error definition may fall is:

- 2xx: Success Codes
- 3xx: Protocol Violation Error Codes
- 4xx: Content Error Codes
- 5xx: Message Processing Error Codes (Permanent Errors)
- 6xx: Message Processing Error Codes (Temporary Errors)
- 7xx: Session Error Codes
- 9xx: Experimental Success and Error Codes

Implementations may experiment with new codes and new features by employing codes values in the 9xx series without fear of interfering with future versions of the WCTP specification. The 900 series codes will never be specified in an officially released WCTP specification. Presumably, once an experimental implementation is deemed to be operational, the developer of this new capability may submit a request to the appropriate WCTP Standards Committee to request that an official status or error code designation be allocated.

The Message Processing (Permanent Errors) codes are error conditions that will not change if the WCTP operation is retried. The Message Processing (Temporary Errors) codes represent error conditions of a temporary nature. A re-submission of the WCTP operation at a later time may result in a successful operation.

3 WCTP Protocol Procedure

The procedure for a WCTP transaction is a request response pair. The actor that is playing the role of a client initiates a WCTP transaction by submitting an operation. The actor that is playing the role of a server provides a WCTP response.

WCTP may be delivered over many different transport mechanisms, but the only one that is defined is HTTP.

3.1 Using HTTP

There are two primary versions of HTTP in use on the Internet as of the writing of this specification. They are HTTP/1.0 and HTTP/1.1 (RFC 1945 and RFC 2068). The most significant difference between these two versions is that HTTP/1.1 allows the TCP/IP connection to be kept open for additional HTTP data transmission.

There can be a significant time delay involved in establishing (and cleaning up) the HTTP/1.0 connection required to transport WCTP packets. This is relevant to actors playing the role of server and actors playing the role of client. Since it is beyond the scope of WCTP to attempt to decrease these delays, the only obvious alternative is to reduce the number and frequency of these connections. HTTP/1.0 may be used to transport WCTP and only one request response pair may be delivered and received per TCP/IP connection. HTTP/1.1 will allow multiple sequential transactions in the same TCP/IP connection.

3.1.1 Multiple Asynchronous HTTP Requests Per Connection

Under HTTP/1.1, a client may submit multiple requests to a server so long as neither the client nor the server “closes” the connection. This means that the client can submit a request even if there are outstanding requests for which responses have not yet been received (asynchronously). Alternately, the client can submit a request, wait for the response, and then submit the next request, and so on. In either case the server must respond to the requests in the order in which the client submitted them as the HTTP specification requires.

Using this model, a WCTP client could reasonably expect to submit multiple requests (XML documents) and receive multiple responses in a single HTTP connection/session. The client can associate each response with its original request, since RFC2068 requires the responses to be in “request order”. This relieves any requirement to add “key fields” to map responses to requests. If the server chooses to implement a “one HTTP request per connection” model using HTTP/1.1 (essentially a “max number of requests per connection” of 1), the server must refuse all requests after the first using the model defined in Section 3.1.2.

3.1.2 Refusing Requests

The largest single issue with this model occurs when the client submits multiple requests and the server has the need to terminate the connection (the client has exceeded some limit on connection time or number of requests). In this case, the server must accept the offending HTTP requests without submitting them for internal processing. Once all acceptable requests have been processed and appropriate WCTP responses have been returned, the server must generate an appropriate (depending on the request type) WCTP/XML failure for the first offending request. In that failure response, the server must indicate a reason code between 700 and 799 (indicating a connection termination for the appropriate reason), and return that failure response along with HTTP “Connection: close”. After this response has been returned, the server closes the connection. No responses will be returned for any other offending requests.

3.2 WCTP Framework

The WCTP protocol defines a number of different control blocks that are transferred between a wire line system and an entry point, or gateway, into the wireless network. Such a gateway understands the wireless network specifics and the native protocols used to communicate within those networks. It is a function of the gateway to map those specifics into the WCTP standardized representations in order to provide uniform appearances regardless of the wireless network over which data is being carried. The gateway "does what is necessary" to provide this mapping. This may include the maintenance of transactional information by the gateway through the "lifetime" of certain messages in order to make up for the inability of certain specific networks to carry all of the information provided within WCTP. The gateway creates WCTP messages on behalf of the wireless device or on behalf of the wireless network reporting a significant event regarding the message delivery process.

Each WCTP control block is referred to as a WCTP Operation. The Operation defines a function being requested or a response being returned. The Operation also carries mandatory and optional parameters associated with it. WCTP Operations are represented in an XML format as dictated by the format language syntax or Document Type Definition (DTD) described in Appendix A WCTP Document Type Definition.

A WCTP Operation may be initiated by either a wire line system or a wireless gateway. An Operation may result in one, many or no direct responses to it.

WCTP is a request/response type protocol, and as such, some WCTP Operations represent protocol requests, and others represent protocol responses. Full correlation of WCTP request and response Operations is presented in 3.3 Actors and Roles.

3.3 Actors and Roles

There are primarily five **actors** associated with WCTP.

- carrier gateway
- enterprise host
- polling enterprise
- transient client
- proxy server

Each of these actors may serve in the role of either a **client** or **server**. In the **client** role, the actor opens a WCTP connection to a server, makes a request, and receives a response back. In the role of a **server**, the actor accepts a WCTP connection from a client, receives a request, and returns a response back to the client. The following sections describe each of the actors, their roles, and interactions with other actors.

3.3.1 Carrier Gateway

The **carrier gateway** represents carrier network and offers access to one or more wireless devices, providing a way to send messages to the devices, to receive device messages or replies if they are available, to obtain device or message status, and to request device capabilities. The **carrier gateway** acts as both a client and a server and is typically the entity with which all of the other actors interact.

The **carrier gateway** accepts requests from external clients and connects to servers to POST status and messages sent from messaging devices. The WCTP requests that a **carrier gateway** may process as either a client or server are shown in Table 1 below:

Request	**Server (receives)**	**Client (submits)**
wctp-ClientQuery	Yes	**No**
wctp-LookupSubscriber	Yes	**No**
wctp-LookupResponse	**No**	Yes
wctp-DeviceLocation	Yes	No
wctp-DeviceLocationResponse	No	Yes
wctp-MessageReply	Yes	Yes
wctp-PollForMessages	Yes	**No**
wctp-ReturnToSvc	Yes	**No**
wctp-SendMsgMulti	Yes	**No**
wctp-StatusInfo	**Yes**	Yes
wctp-SubmitClientMessage	Yes	**No**
wctp-SubmitRequest	Yes	Yes
wctp-VersionQuery	Yes	Yes

Table 1 - Requests processed by Carrier Gateways

3.3.2 Enterprise Host

An **enterprise host** represents a business unit and acts on behalf of one or more message senders and/or message processing applications as both a client and a server. It accepts POSTed messages from messaging devices (sent by way of a **carrier gateway**) and connects to **carrier gateways** to POST messages for delivery to wireless devices.

An **enterprise host** when acting as a server represents access to one or more applications or individuals that receive information from or about wireless devices including the status of a message sent to a device, information about a wireless device's capabilities, or messages from a device. Wireless devices can send unsolicited messages to enterprise servers by providing identifying information in the recipient field. The accepted method for an **enterprise host** to receive messages is via a POST using the HTTP transport protocol. The **carrier gateway** will normally push status, message replies and unsolicited messages to the **enterprise host** as soon as such information is reasonably available.

An **enterprise host** is required to have a permanent connection to the Internet, and when acting as a client, must provide identifying address information in its requests (such as the sender ID when sending a message). This information is used by the **carrier gateway** to post replies and status information messages back to the **enterprise host**.

Figure 1 below shows the relationship between an **enterprise host** and a **carrier gateway**. The arrow indicates the direction of request invocations, and the boxes contain the requests that each actor can make.

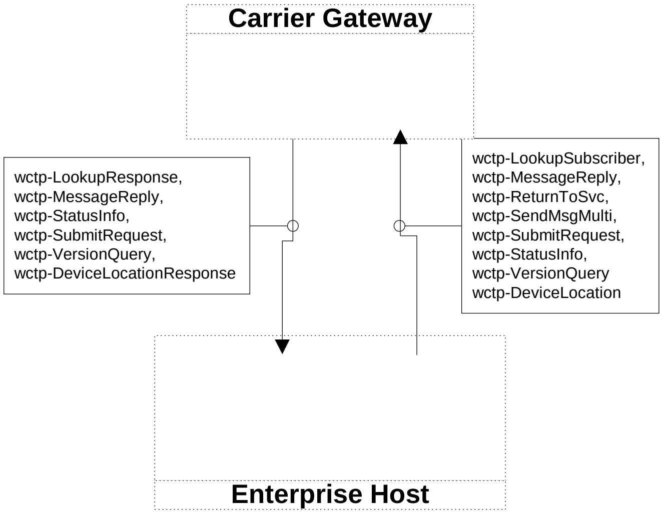

Figure 1 - Enterprise Host and Carrier Gateway Relationship

The WCTP requests that an **enterprise host** may process as either a client or server are shown in Table 2 - Requests processed by Enterprise Hosts.

Request	**Server (receives)**	**Client (submits)**
wctp-ClientQuery	**No**	**No**
wctp-LookupSubscriber	**No**	Yes
wctp-LookupResponse	Yes	**No**
wctp-DeviceLocation	No	Yes
wctp-DeviceLocationResponse	Yes	No
wctp-MessageReply	Yes	Yes
wctp-PollForMessages	**No**	**No**
wctp-ReturnToSvc	**No**	Yes
wctp-SendMsgMulti	**No**	Yes
wctp-StatusInfo	Yes	Yes
wctp-SubmitClientMessage	**No**	**No**
wctp-SubmitRequest	Yes	Yes
wctp-VersionQuery	Yes	Yes

Table 2 - Requests processed by Enterprise Hosts

3.3.3 Polling Enterprise

A **polling enterprise** represents a business unit and acts on behalf of one or more message senders and/or message processing applications as a client and only as a client. It connects to **carrier gateways** to POST messages for delivery to wireless devices and to POLL (`wctp-PollForMessages`) for messages from messaging devices

A **polling enterprise** retrieves replies and status information messages using the `wctp-PollForMessages` operation. It must poll the server regularly for any messages or status information, rather than having the information pushed to them as soon as it is available. Since the `wctp-PollForMessages` operation involves repeated polling, it can generate potentially unacceptable levels of WCTP traffic. Because of this, the `wctp-PollForMessages` operation has a built-in throttle whereby the **carrier gateway** can instruct a **polling enterprise** to delay the next poll for a specific period of time. This limitation restricts the total number of messages that can be retrieved per period of time.

A **polling enterprise** is not required to have a permanent connection to the Internet. It must provide identifying address information (a registered poller ID) in its requests (such as the sender ID when sending a message or the poller ID used when retrieving messages). This information is used to retrieve replies and status information messages using the `wctp-PollForMessages` operation.

Figure 2 - Polling Enterprise Host and Carrier Gateway Relationship below shows the relationship between a **polling enterprise** and a **carrier gateway**. The arrow indicates the direction of request invocations, and the box contains the requests that a **polling enterprise** can make. The WCTP requests that a **polling enterprise** host may process as a client are shown in Table 3 - Requests processed by Polling Enterprise Hosts:

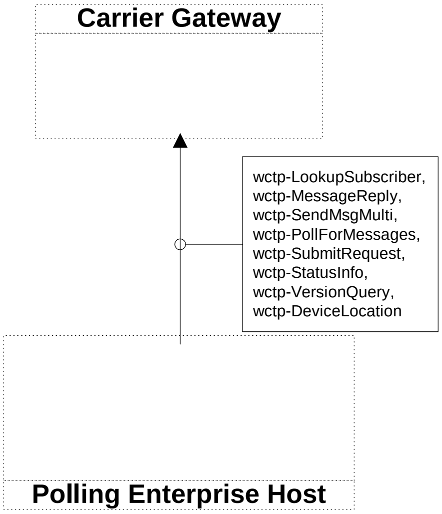

Figure 2 - Polling Enterprise Host and Carrier Gateway Relationship

Request	**Client (submits)**
wctp-ClientQuery	**No**
wctp-LookupSubscriber	Yes
wctp-LookupResponse	**No**
wctp-DeviceLocation	Yes
wctp-DeviceLocationResponse	No
wctp-MessageReply	Yes
wctp-PollForMessages	Yes
wctp-ReturnToSvc	**No**
wctp-SendMsgMulti	Yes
wctp-StatusInfo	Yes
wctp-SubmitClientMessage	**No**
wctp-SubmitRequest	Yes
wctp-VersionQuery	Yes

Table 3 - Requests processed by Polling Enterprise Hosts

3.3.4 Transient Client

A **transient client** typically represents an individual, acting on behalf of a desktop application. It connects to **carrier gateways** to POST messages for delivery to wireless devices and to POLL (`wctp-ClientQuery`) for status or replies to each POSTed message.

A **transient client** must regularly POLL the **carrier gateway** using the `wctp-ClientQuery` operation to obtain status or replies to POSTed messages. Since the polling operation involves repeated polling, it can generate potentially unacceptable levels of WCTP traffic. Because of this, it has a built-in throttle whereby a **carrier gateway** can instruct a **transient client** to delay the next poll for a specific period of time. This limitation, together with the fact that a separate `wctp-ClientQuery` must be executed for each message, restricts the total amount of status and reply information that can be retrieved per period of time.

Transient clients are not expected to have permanent connections to the Internet, although they may. **Transient clients** are anonymous, and any identifying information in the requests they send to a server is not validated or used for delivery (such as the sender ID when sending a message). Because **transient clients** are anonymous and there is no address available for a device to use as a recipient, they cannot receive unsolicited messages from wireless devices.

Figure 3 below shows the relationship between a **transient client** and a **carrier gateway**. The arrow indicates the direction of request invocation, and the box contains the requests that a **transient client** can make.

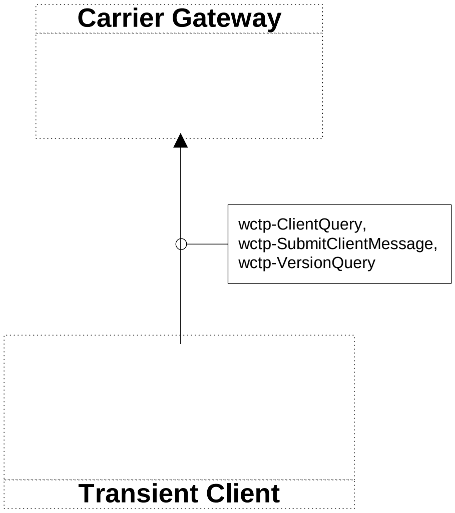

Figure 3 – Transient Client and Carrier Gateway Relationship

The WCTP requests that a **transient client** may process are shown in Table 4:

Request	**Client (submits)**
wctp-ClientQuery	Yes
wctp-LookupSubscriber	**No**
wctp-LookupResponse	**No**
wctp-DeviceLocation	No
wctp-DeviceLocationResponse	No
wctp-MessageReply	**No**
wctp-PollForMessages	**No**
wctp-ReturnToSvc	**No**
wctp-SendMsgMulti	**No**
wctp-StatusInfo	**No**
wctp-SubmitClientMessage	Yes
wctp-SubmitRequest	**No**
wctp-VersionQuery	Yes

Table 4 - Requests processed by Transient Clients

3.3.5 Proxy Server

A **proxy server** serves as an intermediary between a **carrier gateway** and at least one of the other types of actors. The **proxy server** appears as a **carrier gateway** to its clients (**enterprise host**, **polling enterprise**, or **transient client**) and appears ~~at~~ as an **enterprise host** to a **carrier gateway.**

Figure 4 below shows the relationship between a **transient client** and a **carrier gateway** when a **proxy server** is involved. The arrow indicates the direction of request invocation, and the boxes contain the requests that each actor can make.

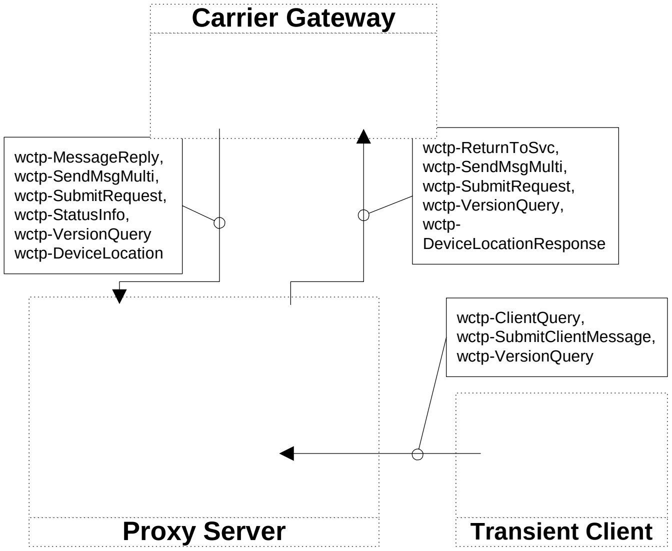

Figure 4 - Transient Client and Carrier Gateway Relationship with Proxy

Proxy servers may be used to accommodate security concerns by placement in a DMZ where a carrier gateway can POST unsolicited messages and solicited status messages to the proxy server. A polling enterprise may then POLL the proxy server for the messages and status information. This configuration allows a business entity to satisfy security goals by not opening holes in its firewall and satisfy performance goals since they are not constrained by any throttling provided on a carrier server.

A carrier may choose to use a proxy server to offload traffic by transient clients and polling enterprises from its carrier gateway. It is beyond the scope of this document to define and describe all applications and usage of proxy servers.

3.4 WCTP Operation Declarations

A WCTP operation must begin with a suitable XML document type definition declaration such as:

```
<?xml version="1.0"?>
  <!DOCTYPE wctp-Operation
    SYSTEM "http://dtd.wctp.org/wctp-dtd-v1r1.dtd">
```

Any alternate form of the above XML declaration is acceptable so long as it meets the following criteria:

- The XML version must be 1.0.
- The DOCTYPE must be `wctp-Operation`.

The SYSTEM keyword must appear, and must be followed by a valid URI that will return a WCTP/XML DTD that defines the version of the WCTP protocol being used to create the operation.

3.5 XML Encoding

The only XML encoding type supported by WCTP is UTF-8, which is the default encoding value if the encoding keyword is not specified. XML/WCTP information is further restricted to single octet UTF-8 (corresponding to the set of 7-bit US-ASCII characters), including the #PCDATA in the wctp-Alphanumeric and wctp-TransparentData elements. Binary data can only be passed in the wctp-TransparentData element by using the base64-encoding algorithm.

The following XML declaration is valid and is equivalent to the declaration shown in Section 3.4.

```
<?xml version="1.0" encoding “UTF-8”?>
<!DOCTYPE wctp-Operation
  SYSTEM "http://dtd.wctp.org/wctp-dtd-v1r1.dtd">
```

Although the following is valid by XML standards, and UTF-16 is a valid XML encoding type, it is not valid WCTP.

```
<?xml version="1.0" encoding “UTF-16”?>
<!DOCTYPE wctp-Operation
  SYSTEM "http://dtd.wctp.org/wctp-dtd-v1r1.dtd">
```

3.6 WCTP Protocol Version

Each `wctp-Operation` contains an attribute (`wctpVersion`) that specifies the name of the protocol version being used. Under WCTP 1.1, the only valid `wctpVersion` value is “`WCTP-DTD-V1R1`”. For future releases of WCTP, version names will be of the form WCTP-[SUBTYPE]-V<major release number>R<minor release number>. Version 2.0 is expected to implement Subtypes other than DTD. Since the valid values for the `wctpVersion` attribute were not defined prior to this release, any unrecognized values will be treated as referring to WCTP version 1.0 (as though the user had entered “`WCTP-DTD-V1R0`”).

Carriers are allowed to implement protocol extensions to support operations that are not yet included in WCTP. These carrier-specific extensions must have version names of the format “<PROTOCOL>-<SUBTYPE>-V<X>R<Y>”. The <PROTOCOL> value may be any all-capital name **other than** WCTP, and the <SUBTYPE> may be any all-capital value so long as the total length in characters of the version name is no more than 32. The <X> and <Y> values refer to the major and minor version numbers as mentioned above.

The name of the WCTP Document Type Definition (DTD) file being used to define the operations appears in the SYSTEM specification as shown in Section 3.4 above. The version name may be reflected in the DTD filename. If it is, the filename will be of the form **wctp-dtd-v**<major release number>**r**<minor release number>**.dtd**. For example, `wctp-dtd-v1r2.dtd` is the official file name associated with this release of WCTP.

3.7 WCTP Tokens

Each `wctp-Operation` returned by a WCTP server can contain an optional attribute (`wctpToken`) specifying that server's "current" configuration status. Clients and proxies should examine these tokens to detect if the WCTP server has changed any aspects of its support (operations, configuration, or version information). As of this release, clients only read tokens and do not generate them.

As an example of using the `wctpToken` attribute, consider when a client receives a `wctp-Confirmation` containing:

```
wctpToken="1AA"
```

when he had previously received:

```
wctpToken="11AA"
```

The client then knows that some aspect of the server's support has changed, and that he should perform a version query operation to determine what has changed.

3.8 Call Flow Between Network Elements

The operations specified above are used to convey information between the wire line and wireless network and possibly between elements within the wireless network itself. **Figure 5** shows several call flows between the enterprise host and other network elements. The diagram assumes that messages from the wireless network that need to be returned to the enterprise host are being pushed directly to the host using the recommended HTTP POST methodology.

The call flows are read from the top of the diagram down showing the types of operations that may be moving between elements in normal and error cases. The diagram also shows cases in which the user of a wireless device is responding to a specific message as well as a case in which an unsolicited message is generated from the wireless device. It should be pointed out that some wireless two-way technologies support the direct association of a message response against a specific message while some other wireless devices may send such responses as new, seemingly unsolicited messages that are not tagged to the original message. Where the device itself is not capable of tagging subsequent responses, the method of associating transmissions and responses are outside of the scope of the WCTP protocol.

Figure 5 helps to point out several items for the WCTP application developer on the wire line side as well as to the infrastructure equipment suppliers on the wireless network side. The definitions of the WCTP control blocks (operations) themselves do not make it apparent how a call flow may proceed. In fact, the sequence of operations that may occur could be dependent upon the nature of the transport protocol being utilized as explained in the following paragraphs.

Referring to the figure, it is shown that when an operation is submitted (set #1), it is possible to get an XML formatted error response posted back. The Submit operation does carry optional parameters (referred to as *notifyWhen* type parameters) that indicate the requirement to post back one or more delivery status conditions to the message submitter. When the requested status condition occurs, it will be queued for return to the submitter. The "Acknowledged" status shown in the second Submit in the call flow (set #2) is not one that it optionally requested. This status return is provided for implementations in which an error or status response may not be immediately available to return to the submitter, but the transport protocol being used requires a return response before closing the connection. This is the case of the submit/response HTTP protocol that is used as the primary WCTP transport protocol. The second Submit operation shown in the figure (as set #2) is assumed to be operating over an HTTP type connection. Although the Submit request received was accepted by the Internet Gateway element of the wireless network, the diagram is attempting to show the generalized case in which the operation may be determined to be un-executable or in error at some later point in its processing (such as when it is processed by the WCG element within the wireless network). That determination may not be able to be made in a "timely" fashion and it may not be desirable to leave HTTP transport connections open for longer than they need to be. The "Acknowledged" response (success code 200 of Appendix E Error and Success Codes) is returned via `wctp-Confirmation` in order to acknowledge the receipt of a valid WCTP operation that it may or may not be able to successfully complete.

From the point of view of the creator of a WCTP application, it is only necessary to understand that an error response to a submitted request may be returned to the application some time in the future. A received response status is only an intermediate response condition and does not necessarily mean that the operation was acceptable for execution.

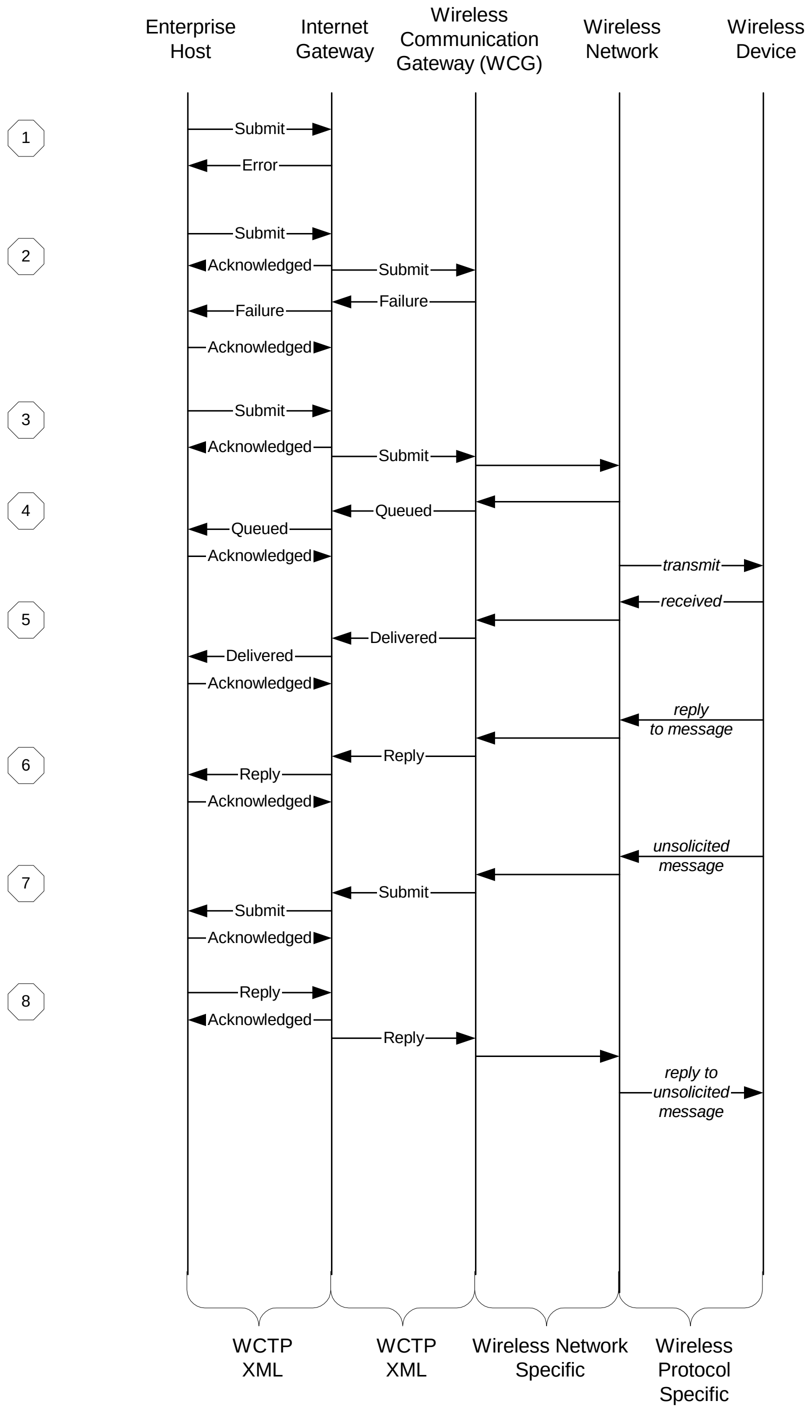

Figure 5 - Call Flow Between Network Components

The third Submit operation in the figure (set #3) shows where a valid Submit operation was presented. When the message being submitted is validated, properly queued and safe-stored for delivery within a network specific element of the wireless Network, a "Message Queued" delivery status may be returned to the message sender (set #4) to indicate that the requested operation is pending. The diagram shows the Queued Status being pushed to the Enterprise host through the HTTP protocol. As required by HTTP, a response must be sent to the pushed Queued status. That response is sent as a WCTP Acknowledgement operation (`wctp-Confirmation`) that confirms the receipt of the pushed data.

The example shows the pushing of a Delivered Status notification (set #5) upon the successful delivery of the message to the wireless device because the Submit operation had requested a "*notifyWhenDelivered*" status report. When the user of the wireless device replies to the received message, a WCTP MessageReply type operation (set #6) is pushed to the Enterprise Host.

The example also shows the case where a wireless device initiates an unsolicited message (set #7). This type of message is pushed to the Enterprise Host as a WCTP Submit operation initiated by the wireless network. The Enterprise host may send a response to the unsolicited message (set #8) by initiating a MessageReply operation against the messageID assigned by the wireless network.

4 Addressing in WCTP

The address format in WCTP is designed to allow maximum flexibility for developers on the wire line and wireless platforms. Many developers will be familiar with addressing via the Internet for host-to-host communications. Some devices also have specific routing mechanisms that may exist in the payload or external to the payload. There may also be special addressing rules or instructions that apply to a particular class of devices or wireless networks. The carrier or device manufacturers should be approached in these cases if addressing documentation is not readily available.

4.1 General Address Identification in WCTP

Address identifiers in WCTP are presented in the following general form:

[entity-address@]transport-address[1]

transport-address is used by the WCTP transport layer to deliver WCTP operation to a carrier gateway or an enterprise host.

A WCTP server or an enterprise host uses an optional **entity-address** to identify the ultimate source or destination of an operation. It can represent a wireless device, a user name, an application, or it can carry any other carrier or host-specific meaning.

The character set for both transport- and entity-address portions must conform to RFC 2396, section 2 URI Characters.

4.1.1 Transport-address

transport-address has the following general form:

[Protocol**:**][//]Domain[**:**Port][/Path]

The optional **Protocol:** is used to identify the alternative transport for WCTP. If no Protocol specified, the default “HTTP” is assumed and means standard WCTP transport over HTTP. Other protocols may be supported, for example, “SMTP:” or “TCP:” might be used. In order for an alternate transport protocol to be used, the WCTP client and server must both support it.

Domain is used to locate the carrier gateway or the enterprise host where WCTP operation is being delivered. In the case of a delivery to a polling enterprise, the domain portion is used to present the `pollerID` (as defined in section 4.1.3 PollerID) so that any response or requested status information is held for a polling enterprise.

In push operations the domain value of a carrier gateway or an enterprise host should present a fully qualified domain name resolvable via DNS or an IP address, but the following set of rules should be used to reduce the length of the complete address.

If no forward-slash "/" is present (the Path portion of the address is not specified), the default path **/wctp** must be assumed. It should be noted that specifying a path of "/" (the root directory) qualifies as a valid path, and /wctp will NOT be assumed.

[1] While in most cases WCTP address identifiers appear similar to either e-mail identifiers of RFC 822 or URL/URI identifiers of RFCs 1738 and 2396, the full general address specification form in WCTP is not compliant with any one of the mentioned RFCs.

The prefix "**wctp.**" must be prepended to a valid domain under most conditions. Many domains are expected to co-locate a WCTP gateway on the same Web server, but as a separate sub-domain, e.g. wctp.carrier.net.

- If an IP address is provided, then "**wctp.**" must not be prepended to it.
- If the domain is preceded by "**//**", then "**wctp.**" must not be prepended to the domain.
- If "**wctp.**" is already prepended to a valid domain, then "**wctp.**" must not be prepended again.

Both the optional **Port** and **Path** contain additional routing information for the WCTP transport layer.

If no port is provided (the Port portion of the address, together with the colon ":", is not specified), the default port of **:80** must be assumed.

The following examples in Table 5 are intended to illustrate and clarify these concepts, and show the Domain, Path, and Port that must actually be used for POSTing:

Transport Address	**Domain**	**Path**	**Port**	**Delivery Address**
123.45.78.90	123.45.78.90	/wctp	80	123.45.78.90:80/wctp
123.45.78.90:123/	123.45.78.90	/	123	123.45.78.90:123/
wctp.full.com/appl	wctp.full.com	/appl	80	wctp.full.com:80/appl
//asis.com/appl	asis.com	/appl	80	asis.com:80/appl
//asis.com	asis.com	/wctp	80	asis.com:80/wctp
asis.com	wctp.asis.com	/wctp	80	wctp.asis.com:80/wctp
full.com/appl	wctp.full.com	/appl	80	wctp.full.com:80/appl
full.com:445/appl	wctp.full.com	/appl	445	wctp.full.com:445/appl

Table 5 - Examples of Delivery Addresses

4.1.2 Entity-address

entity-address has the following general form:

[Scheme**:**]Entity[**:**Port][/Path]

Entity as well as optional **Port** and **Path** present identification and possibly additional routing information meaningful to the corresponding side of WCTP communication.

Optional **Scheme** component may be used for the specification of entity identification method and is compliant with a notion of scheme from RFC 2396. Scheme, if specified, can be used by infrastructure of wireless network or an enterprise to distinguish and properly route WCTP traffic. In the absence of a specified scheme, “wctp:” is assumed by default.

The optional portions of the entity-address format are normally used to assist with application level routing on a device when application routing is outside the message payload. These fields can be used to further assist in payload routing for an enterprise application as well.

4.1.3 PollerID

A **pollerID** is a string that identifies an address as belonging to a polling enterprise.

The `pollerID` is presented as the domain portion of an address. Therefore, a valid `pollerID` must contain at least one ".". This condition is used to insure that the same character string is not defined as a `pollerID` and as an alias (as defined in section 4.1.4 Alias).

Assignment and maintenance of a `pollerID` is specific to a particular carrier gateway implementation and is beyond the scope of this specification.

4.1.4 Alias

An **alias** is a string of characters that is recognized by a carrier gateway as representing an enterprise host address. Benefits of using an alias includes

- Decreasing the number of characters that a device sends over the air by replacing the address with the shorter alias string.
- Easier management of WCTP address changes by limiting updates of the changes to alias registration at the carrier gateway rather than pushing changes to wireless devices.

4.1.4.1 Carrier Alias

A **carrier alias** is an alias defined only for use with a particular carrier gateway. Other carriers will not recognize a carrier alias unless it has also been registered with the other carrier gateways. Registration and maintenance of carrier aliases is specific to each carrier gateway and is beyond the scope of this specification. An enterprise may use a carrier alias if

- Decreasing the number of characters sent off the air is important.
- The URI used is expected to change.
- Traffic will be limited to a single carrier's devices.

4.1.4.2 Global Alias

Alternatively, a **global alias** is an alias that is recognized by all cooperating carriers as resolving to the same WCTP address. Use of a global alias prevents an enterprise from having to register an alias to multiple carrier gateways. Global aliases offer the following benefits

- Single point of alias registration for all cooperating gateways.
- Guaranteed unique alias for all cooperating gateways.

At this time an entity for managing global aliases has not been established. Until this entity has been defined, global aliases will not be available.

4.1.4.3 Attributes

An alias may be used to resolve a WCTP address for the following attributes

- Recipient address when used by a wireless device to send to a wire-line application.

- `senderID` when using the `wctp-SubmitRequest`, `wctp-SendMsgMulti`, or `wctp-MessageReply` operations.
- `sendResponseToID` when using the `wctp-SubmitRequest`, `wctp-SendMsgMulti`, or `wctpMessageReply` operations.

4.1.4.4 Format

An alias must contain at least one alpha character. The first character of the alias will determine if it is a carrier alias or a global alias, with an exclamation point ('!') indicating a carrier alias.

4.1.4.5 Usage

An alias may represent either the entire WCTP address or the entire WCTP transport address.

The replacement value of an alias representing the entire WCTP address is as follows:

[entity@][protocol:][//]domain[:port][/path].

The replacement value of an alias representing the entire WCTP transport address is as follows:

[protocol:][//]domain[:port][/path].

An entity may be specified with an alias that has been defined only as the transport address. It is invalid however to specify an entity with an alias that has been defined as a WCTP address that includes an entity specification. Examples and usages are shown in Tables 6, 7 and 8.

4.1.4.6 Examples

Alias	**Type**	**Replace Value**	**Resolves To**
Acme	Global	wctp.acme.com:80/app	wctp.acme.com:80/app
Asis	Global	asis.com	asis.com:80
!pacme	Carrier	person@acme.com/wctp	person@acme.com:80/wctp
Bacme	Global	bob@acme.com	bob@acme.com:80

Table 6 - Valid Aliases

Recipient address	**Alias**	**Replacement Value**	**Resolves To**
Acme	Acme	acme.com	acme.com:80
bob@acme	Acme	acme.com	bob@acme.com:80
Bacme	Bacme	bob@acme.com	bob@acme.com:80

Table 7 - Valid Usage

Recipient Address	Alias	Replacement Value	Explanation
http://acme	acme	acme.com	Alias must be entire WCTP or Transport Address
wctp.acme/wctp	acme	acme.com	Alias must be entire WCTP or Transport Address
bob@bacme	bacme	bob@acme.com	This would resolve to bob@bob@acme.com:80 which is invalid.

Table 8 - Invalid Usage

4.1.5 WCTP Addresses for Wireless Devices

For wireless devices, the Entity portion of a WCTP identifier can represent a numeric or non-numeric identifier of a wireless device. In this case, Port and Path parts of entity-address are relative to the wireless device id, and may specify delivery information for an application running on a wireless device.

Examples:

- 1234567@abcwireless.com
- 8005551212.98765@abcwireless.com
- 5551212:563@abcwireless.com
- john_doe@abcwireless.com
- john_doe/chess_app@abcwireless.com

4.1.6 WCTP Addresses for Wire line Hosts

In the case of enterprise hosts or polling enterprises, the use of entity, port and path components of the entity-address is particular to the implementation of the enterprise host or polling enterprise.

The entity can represent a user name, an application name or carry any other enterprise host or polling enterprise specific meaning. The port can be used to override ~~of~~ the default value of port 80. The path can signify the location of particular applications software on the server to process the WCTP operation being sent or can be used to pass additional information to the WCTP application. This specification does not define the standard behavior or representation for these components.

Examples:

- inventory@myenterprise.com
- joe.user@https:myenterprise.com/wctp-gateway
- accounting/receivables@http://myenterprise.com:8080/process-wctp
- johndoe@im-gateway.com

4.1.7 Identifiers For Transient Clients

The format of identifiers for transient clients in WCTP is not fixed. Normally, uniqueness of such an identifier cannot be guaranteed. Therefore, wireless systems cannot rely on the `senderID` for messages submitted by transient clients. In case of transient clients, for delivery of responses the wireless system must rely on the uniqueness of an assigned tracking number or a combination of tracking number and recipient identifier of the original message. The `senderID`, specified on a submitted message by a transient client, must be specified for authentication when querying for status.

4.1.8 Message Response Redirection to non-WCTP Destination

WCTP allows specification where the responses to a particular message are to be sent to an alternate address via the `sendResponsesToID` attribute of `wctp-MessageControl`. Wireless Carriers may support redirection of responses to non-WCTP destinations, like e-mail, faxes, phones, etc. To support this feature WCTP reserves the following prefixes that can be specified in `sendResponsesToID`: “**mailto:**”,” **faxto:**”, and “**phoneto:**”.

5 WCTP Date and Time Formatting

5.1 Date and Time Formats

This section describes the date and time format used by WCTP in all contexts where a date that can be parsed is required. The format shown here is a selected profile of options from ISO8601:1988 (with Technical Corrigendum 1 applied), hereinafter referred to as ISO8601.

5.1.1 WCTP Time Interval Format

The format for a period of time (such as `minNextPollInterval` in the `wctp-PollResponse`) is:

"x"

where x is any integer value, and the units are "seconds". An example using the minNextPollInterval attribute of the `wctp-PollResponse` would be:

```
minNextPollInterval="3600"
```

to represent a requested delay of 3600 seconds, or one hour, before the user submits the next `wctp-PollForMessages` for the `pollerID` used in the request.

5.1.2 WCTP Date Format

The format for a date string in WCTP Date Format is:

CCYY-MM-DD

Where CCYY is the four-digit year (century and year, as described in ISO8601), MM is a two-digit month number, DD is the two-digit ordinal number of the day within the calendar month, and the separator character is a "hyphen" ("-"). This format is the *Extended Format* described in ISO8601 Section 5.2.1.1 with the separator as described in ISO8601 Section 4.4. WCTP implementations must use this format for all date strings specified as being in WCTP Date Format.

Section 5.1.4 denotes the format of the string for specifying both date and time.

5.1.3 WCTP Time Format

The format for a time string in WCTP Time Format is:

HH:MM:SS

Where HH is the two-digit hour in 24 hour notation ranging from 00 to 24 (this is not a typo), MM is the two-digit minute ranging from 00 to 59, SS is the two-digit seconds ranging from 00 to 59, and the separator character is a "colon" (:). This format is the Extended Format described in ISO8601 Section 5.3.1.1 with the separator as described in ISO8601 Section 4.4. WCTP implementations must use this format for all time strings specified as being in WCTP Time Format.

Note that midnight has two representations:

```
00:00:00
```

```
24:00:00
```

This is deliberate and in accordance with ISO8601 Section 5.3.2.

5.1.3.1 Sub-second Resolution

WCTP Time Format for representing sub-second granularity follows ISO8601 Section 5.3.1.3 and thus uses a "comma" (,) separator and an arbitrary number of digits representing the fraction down to the appropriate level of precision. Thus, the format for time with sub-second resolution is:

HH:MM:SS,S

Where the "comma" (,) is a literal character (ISO8601 separator) and "s" after the comma is "to the right of the decimal mark" and indicates the sub second value. The number of digits in the sub-second value, the precision of the sub-second value, and the ability of a given implementation to honor that precision are quality of implementation issues and are not specified by WCTP.

5.1.3.2 Time Zones

All times specified within WCTP must be specified using Universal Coordinated Time (UTC), previously known as Greenwich Mean Time (GMT). Implementations are expected to translate these times into the appropriate local time presentation format before utilizing the time in the manner in which it is being requested.

Although ISO8601 Section 5.3.3 requires that a literal character “Z” (time zone designator) be appended to the above format string to indicate UTC, WCTP does not support the use of suffix “Z” and the format always represents UTC time. Including the "Z" suffix will result in rejection by a properly implemented WCTP gateway (see Error 405).

If the user erroneously specifies a local time rather than UTC for a timestamp attribute, such as `submitTimestamp`, when submitting a message, the WCTP gateway will simply pass the timestamp through in the response operation. However, when the user erroneously specifies local time for scheduling attributes (such as `deliveryAfter` or `deliveryBefore`) during message submission, any of the following situations may occur, depending on the offset of the local time from UTC and the scheduling requested:

- The message may be delivered before the `deliveryAfter` time.
- The message may be delivered after the `deliveryBefore` time.
- The message may expire and never be delivered.
- The message may be rejected.

To obtain the proper message delivery schedule, the user must ensure that all timestamps are expressed in UTC, without the “Z” suffix.

5.1.3.3 Time Zone Examples

A user submits a message from New York at 13:30 EST (Eastern Standard Time) on February 26, 2001. The user must specify the `submitTimestamp` attribute of the header (`wctp-SubmitHeader`, `wctp-SubmitClientHeader`, or `wctp-MsgMultiHeader`) as follows (since EST is -5 hours from UTC):

```
submitTimestamp="2001-02-26T18:30:00"
```

A user submits a message from Dallas at 13:30 CDT (Central Daylight Time) on May 26, 2001 to be delivered after 15:00 CDT. The user must specify the `submitTimestamp` attribute of the header (`wctp-SubmitHeader`, `wctp-SubmitClientHeader`, or `wctp-MsgMultiHeader`) and `deliveryAfter` attribute of the control (`wctp-MessageControl`, `wctp-ClientMessageControl`, or `wctp-MsgMultiControl`) as follows (since CDT is -5 hours from UTC):

```
submitTimestamp="2001-05-26T18:30:00"
```

```
deliveryAfter="2001-05-26T20:00:00"
```

No matter what time zone the recipient is in, the message will be delivered after 20:00 UTC.

A user submits a message from San Francisco at 13:30 PST (Pacific Standard Time) on February 26, 2001 to be delivered after 15:00 PST but before 17:00 PST. The user must specify the `submitTimestamp` attribute of the header (`wctp-SubmitHeader`, `wctp-SubmitClientHeader`, or `wctp-MsgMultiHeader`) and the `deliveryAfter` and `deliveryBefore` attributes of the control (`wctp-MessageControl`, `wctp-ClientMessageControl`, or `wctp-MsgMultiControl`) as follows (since PST is -8 hours from UTC):

```
submitTimestamp="2001-02-26T21:30:00"
```

```
deliveryAfter="2001-02-26T23:00:00"
```

```
deliveryBefore="2001-02-27T01:00:00"
```

5.1.4 WCTP Date and Time Format

When date and time need to be specified in a single string, the WCTP Date and Time format is:

CCYY-MM-DDTHH:MM:SS,S

Where "T" (uppercase T) is a literal character [ISO8601 designator]. This format is the Extended Format of calendar date and time of day as described in ISO8601 Section 5.4.1 clause (a).

WCTP operations must not specify invalid combinations of fields such as February 31. WCTP implementations should reject invalid combinations of fields rather than trying to interpret them.

As in section 5.1.3.2 Time Zones, all times expressed as part of the Date and Time format are to be expressed in UTC.

6 WCTP Version Negotiation

As the WCTP protocol grows and evolves, there is a need for clients (applications requesting a WCTP service) and servers (applications providing a WCTP service) to negotiate with each other in order to establish a set of common requests and responses.

In the standard WCTP model, each WCTP gateway has a server application (that receives requests from outside clients) and a client application (that makes requests against other servers). In each of these relationships, the client needs to know what version of the WCTP the server supports before making a request against it.

In both cases, a version query (a request for information about which version(s) of the WCTP protocol is/are supported) lodged against a server should provide the client with information about the DTDs supported by the server. It is then the client's responsibility to use this information to create WCTP requests that are compatible with the server's capabilities.

6.1 WCTP Version Query Operation

The Version Query request (`wctp-VersionQuery`) allows clients to request information about the DTDs that a server application currently supports. In it, an inquirer identifies itself and optionally provides a time-stamp for its request.

The response to a Version Query request is a Version Response (`wctp-VersionResponse`) as shown below in Table 9. If, however, the Version Query received by the server cannot be parsed (invalid XML characters, for example), the response will be a generic Failure.

Request	Response	Comments
VersionQuery	VersionResponse	Version support information is requested and supplied for the currently supported DTDs.
	Failure	See error table.

Table 9 – Version Query Operations

The requester can use the `inquirer` attribute to identify itself to the server. This value is required.

The user can provide a date/time stamp with the query in the `dateTime` field. If this value is provided, it must be returned by the server in the `dateTimeOfReq` field in the `wctp-VersionResponse` element. This value is a key for connecting the response with the original request.

6.2 Version Response Operation

The Version Response (`wctp-VersionResponse`) is sent to users in response to a `wctp-VersionQuery` request. It is not an initiating operation (a request). Figure 6 below shows the structure of the Version Response.

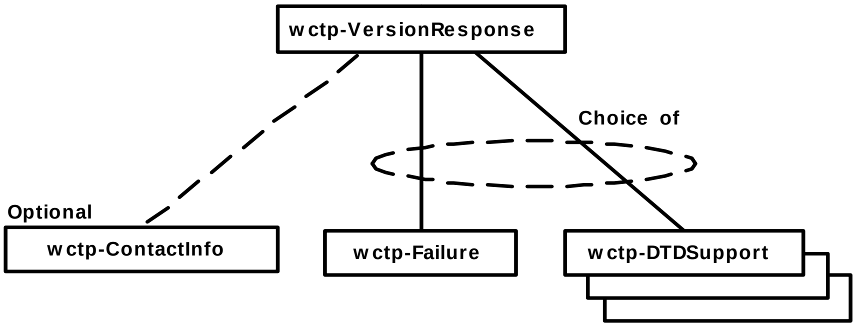

Figure 6 – wctp-VersionResponse Entity

It may contain the following WCTP errors (see Table 10 below). For details on errors in the range from 300 to 899, see Appendix E Error and Success Codes.

Code	Description	Code	Description
0	Undefined Error	302	XML Validation Error
200	Success	303	Version Error
300	Operation Not Supported	400	Function Not Supported
301	Cannot Parse Input		

Table 10 – Error Codes in Version Response Failure

The value submitted in the `inquirer` attribute of the original `wctp-VersionQuery` will be returned here in the `inquirer` attribute. This attribute is provided as a key for connecting this response to the original request.

If the requester provided a timestamp in the `dateTime` attribute of the original `wctp-VersionQuery`, that value will be returned here in the `dateTimeOfReq` attribute. This attribute is provided as an optional key for connecting this response to the original request.

The server must return information in the `responder` attribute identifying the URI/URL against which this version information is valid. It should be noted that this URI may be different from the originator of the `wctp-VersionQuery`.

The server may also provide a date/time stamp for the response in the `dateTimeOfRsp` field. If this value is provided, it is for informational purposes only. Similar to the inquirer's date-time stamp, this can be used for filing away responder capability information.

The server may optionally return a date/time value in the `invalidAfter` attribute. This date/time value represents the time until which the requester may rely on the response. The responder may alter this date-time at will, however, and different inquirers may obtain different time-stamps for the exact same capabilities set. If there are any doubts as to the validity of the version information, the requester should refresh by again requesting version information via a `wctp-VersionQuery` after the `invalidAfter` date-time.

6.2.1 Contact Information

The `wctp-VersionResponse` may contain an optional `wctp-ContactInfo` element, providing contact information through which the inquirer can obtain assistance and additional information about the server.

The `email` attribute is intended to provide a valid SMTP e-mail address. Any valid messaging address may be supplied.

The `phone` attribute is intended to provide a voice telephone number. Only valid Hayes-type telephone characters may appear: left- and right-parentheses, space, hyphens and commas. No Hayes-type alpha commands, such as ATDT, etc., are allowed. An 'x' may appear at the end of the telephone number to indicate that a phone extension of one or more numeric digits follows. If 'x' is used, then numeric characters may follow up to the limit of the field.

The following are valid examples for phone:

- `“8005552113”`
- `“1-800-555-2368”`
- `“1(800)555-2368”`
- `“(800) 555-2368”`
- `“8005436789x5128”`

The `www` attribute is intended to provide a URL or URI identifying a valid web address that may be used to find additional information about the WCTP support offered by the carrier gateway or enterprise host.

The `info` attribute is intended to provide an additional type of contact other than the explicit types already listed or more specific instructions to contact the appropriate responsible party concerning a WCTP implementation.

6.2.2 Failure

The `wctp-VersionResponse` may contain a `wctp-Failure` element if the version request fails.

6.2.3 DTDSupport

The `wctp-VersionResponse` will contain at least one `wctp-DTDSupport` element if the version request succeeds. A `wctp-DTDSupport` element provides support information about the DTD identified by the name that it contains. Each `wctp-DTDSupport` element returned in the `wctp-VersionResponse` provides version information about the specified DTD as it is implemented in the carrier gateway or enterprise host.

The `dtdName` attribute should have a format consistent with the `wctpVersion` as defined in Section 3.6.

The `verToken` attribute, if provided, is an arbitrary string representing the "current" token for version information, and may be cached long-term. This value must change whenever the specified DTD support has been modified (a new feature or operation is added, a feature or function is dropped, etc.). Any change of the wctpToken value in subsequent WCTP Operations indicates that there has been some change in the overall support for the WCTP protocol-level features or DTDs, and the submitting application must initiate a wctp-VersionQuery. Care should be exercised to assure that no token is ever re-used with a different meaning.

The `exceptions` attribute defines whether the specified supported DTD has exceptions (when some aspect of the DTD is unsupported or only partially supported) or not. If the exceptions attribute is "Yes", then there is at least one feature of the DTD that is not fully supported. In WCTP version 1.2, any additional information about these exceptions must be obtained through the resources identified in one of the contact addresses in the `wctp-ContactInfo` element.

The `supportType` attribute defines the type of support (if any) that is available for the specified DTD. Valid values are shown in Table 11.

`Supported`	operations may be submitted under this DTD (see the exceptions attribute)
`Deprecated`	operations may be submitted under this DTD, but the use of this DTD is discouraged (an alternative may be available, and this DTD may be moved to `NotSupported` at any time)
`NotSupported`	operations submitted under this DTD will be rejected

Table 11 – Support Type Definitions

Additional attributes may be returned to provide a recommendation from the carrier gateway concerning Deprecated or NotSupported versions of the protocol. The `supportUntil` attribute should indicate how much longer a particular version of WCTP would be supported. The `replacement` attribute should indicate the carrier gateways' preferred version of WCTP.

7 Common Elements

Within WCTP there is commonality of unique elements that are consistent irrespective of the actor (carrier gateway, enterprise host, polling enterprise, transient client, or proxy server). Whether the actor is functioning as a client or as server these general elements are constant throughout WCTP. The five components where continuity exists are the Header Element, Payload, Confirmation Response, Response Header, and Notification.

7.1 Header Elements

Operations that send messages contain a header element. They differ slightly but also have many common features. These headers are `wctp-SubmitHeader`, `wctp-MsgMultiHeader`, `wctp-ResponseHeader` and `wctp-SubmitClientHeader` in operations `wctp-SubmitRequest`, `wctpSendMsgMulti`, `wctp-MessageReply`, and `wctp-SubmitClientMessage` respectively.

7.1.1 wctp-Originator

The wctp-Originator element identifies the sender of a message. It contains the `senderID`, which is the return address of the sender. An optional `securityCode` can be used to authenticate a sender if required by a wireless carrier. Another optional field is `miscInfo` that has no specified meaning, but may be used for logging or troubleshooting an application. For any send operation other than `wctp-SubmitClientMessage`, if the `sendResponsesToID` field is not provided in the corresponding Control element, any notifications and replies will be delivered to this address. Note: replies and notifications to a message sent via the `wctp-SubmitClientMessage` can only be accessed using a `wctp-ClientQuery` operation.

7.1.2 wctp-Recipient

This element identifies the recipient of a message. It contains the `recipientID`, which is the address of the recipient of the message. It may also contain `authorizationCode` that is an attribute that a wireless network may require as authentication information for a particular `recipientID`.

7.1.3 submitTimestamp

Header elements contain a `submitTimestamp` attribute that the sender populates with the time and date when the message is originally submitted to the WCTP server. See Section 5 WCTP Date and Time Formatting for more information.

7.1.4 Control

Headers also have a control element with various common parameters. Since the actual parameters vary between the operations, each operation has its own version of the control element. The control elements are `wctp-MessageControl`, `wctp-MsgMultiControl`, `wctp-MessageControl`, and `wctp-ClientMessageControl` for operations `wctp-SubmitRequest`, `wctp-SendMsgMulti`, `wctp-MessageReply`, and `wctp-SubmitClientMessage` respectively.

7.1.4.1 sendResponsesToID

This field conforms to addressing requirements but is an alternative delivery point for notification messages from the WCTP server and/or message replies from the recipient(s).

Note: The actual behavior invoked by this element is implementation specific. In some cases, all responses, including notifications, pertaining to any given message may be returned to the specified address. Some implementations, however, may return only message replies to the specified address while notifications or status messages are returned to the address specified as `senderID` in the `wctp-Originator`. It is important to check with the Carrier or the owner of the Host server for specifics.

7.1.4.2 allowResponse

By setting this attribute to "false," replies to this message are inhibited even if the recipient's device is capable of replying.

7.1.4.3 notifyWhenQueued

This field requests a notification from the delivering network that the message has been accepted and is queued to be delivered. The ultimate WCTP server that delivers the message into another network, normally a wireless network, shall generate a response to the originator (or `sendResponsesToID`, if specified) at or after the time the message has been placed on the outgoing queue to the recipient.

7.1.4.4 notifyWhenDelivered

This field requests a notification from the delivering network that the message has been delivered. Some wireless networks and devices may not be capable of this type of notification. The delivery notification is sent to the originator when the message is acknowledged by the recipient's device as having been delivered.

7.1.4.5 notifyWhenRead

This field requests a notification from the end recipient when the message has been opened and read. Not all wireless devices are capable of generating read notifications, nor are all wireless networks capable of receiving them or processing them. If the recipient and delivering network are capable of generating, receiving and processing read notifications and in turn generate a notification to the delivering WCTP server, then the WCTP server shall generate a notification response to the originator that the message was opened and read.

7.1.4.6 deliveryPriority

This field is an indicator of the priority the originator places on the message. The wireless service provider specifies behavior associated with this attribute and the priority values HIGH, NORMAL, and LOW.

7.1.4.7 deliveryAfter

This field requests that the message be held until the date-time specified and delivered after that time. The recipient at the time of acceptance must be valid. See Section 5 WCTP Date and Time Formatting for more information on date and time formats.

Capability to store a message in the queue is limited and varies by network. Some networks may be able to store a message for delivery as far as a year into the future. Other networks may be capable of no storage at all other than queuing messages for near immediate delivery.

If the `deliveryAfter` time provided by the sender has passed, the sending network may assume immediate delivery or may assume an error and reject the message.

7.1.4.8 deliveryBefore

This field requests that the message be delivered to the recipient on or before the date and time specified. If the message has not been delivered by the expiration specified, it should be removed from the queue and delivery should be canceled.

If the receiving WCTP server does not support the `deliveryBefore` parameter, i.e. cannot discard a message after acceptance, then the response may be a failure or may be a success with warning that the message was accepted for delivery but the `deliveryBefore` parameter is not supported.

See Section 5 WCTP Date and Time Formatting for more information on date and time formats.

7.1.4.9 preformatted

A wireless carrier may apply formatting to alphanumeric messages to reduce the number of characters that are delivered over the air. Some examples of formatting that may be implemented by a wireless carrier include reduction of white space and consolidation of redundant characters. Setting the `preformatted` attribute to “true” may disable formatting of alphanumeric messages, indicating that the message has been formatted by the sender and should not be reformatted by the wireless carrier. The `preformatted` attribute is not applicable to transparent data messages, which are not to be reformatted by the wireless carrier under any conditions.

7.1.4.10 allowTruncation

This field, when “false,” indicates that the entire message payload must be delivered intact to the recipient. If the message is too long for the device to accept, or the message is too long for the network to pass to the device intact, or it violates network policy on message length, then the message must be rejected with a Failure response. A “true” value indicates that the payload may be truncated to fit into the device, to pass through the network, or to conform to network policy.

7.2 Payload

The payload is the message content sent to or received from a wireless device. The three types of payload include wctp-Alphanumeric, wctp-Transparent, and wctp-MCR. Both wctp-Alphanumeric and wctp-MCR are ASCII text payloads, while wctp-Transparent represents binary payload. Figure 7 shows the major XML elements of the wctp-Payload entity.

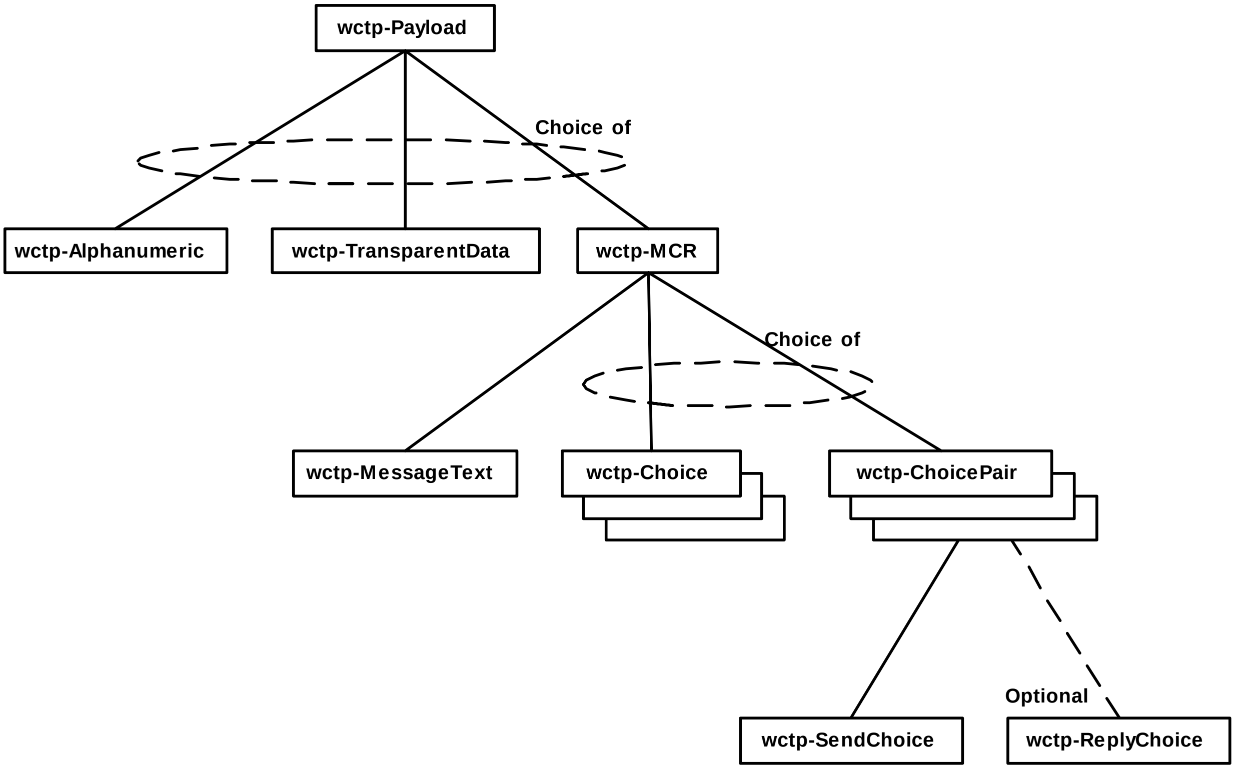

Figure 7 – wctp-Payload Entity

7.2.1 Alphanumeric

An Alphanumeric payload is specified for a device that can display digits or alpha characters that may be human readable. This payload type may contain simply digits in the case of a numeric pager or a cellular phone. Many devices that display alphanumeric characters only support a subset of various character encoding schemes and it is outside the scope of WCTP to standardize this. Some numeric devices only have four bits of character representation. Other devices may have seven bits to represent an alphanumeric message. It is possible to even have full ASCII compliance or a different unique character set displayable on a device.

7.2.2 Multiple Choice Response

MCR (Multiple Choice Response) messages may be sent through all message submission operations. An MCR message has a list of choices to be sent to the device. The MCR response message does permit responses to be returned that are not members of the list of choices. This permits an application to provide a list of "suggested" responses while still permitting the wireless device to return its own specific response. Some wireless devices, such as those that integrate ReFLEX™ technology, provide an over-the-air optimized methodology for supporting this capability. The network then would be able to construct the intended response as a reply to the sender of the original MCR message.

The portion of the message that is initially delivered to the device, which would have the MCR options is delivered in a `wctp-MessageText` element.

The list of choices should contain unique strings of text. If the same text appears within multiple selections, the information provided in the MCR response does not provide sufficient information to know specifically which index of the MCR message was selected. If this information is required, the list of choices must contain unique strings of text. The support and implementation of MCR messages by a wireless device or application is device or application specific.

Other wireless devices could implement the MCR type capability through special application code operating in the wireless device or a computer directly connected to the two-way wireless transceiver. If specialized applications code is used to support the feature, the Wireless network processing the WCTP requests must be aware of the manner in which the capability is implemented in order to properly activate the MCR function of WCTP. WCTP does not support the return of multiple selections at one time.

Submitting a single value to be displayed on the device and returned to the sender as the reply is accomplished by using `<wctp-MCR>` in the payload and by using `<wctp-Choice>.` If the desired behavior is to have one option displayed on the device and a different one returned to the sender, then you should use `<wctp-ChoicePair>` with a `<wctp-SendChoice>` containing the display item and the `<wctp-ReplyChoice>` as the response item. In either case the response `<wctp-MessageReply>` or `<wctp-ClientMessageReply>` to an MCR message should have the `MCRReplyChoice` attribute set to “`true`”.

7.2.3 Transparent Data

Specialized devices or devices with an operating system may require transparent or binary data messages. The facility used by WCTP to enable transparent data payloads is the `wctp-TransparentData` element that consists of PCDATA, a `type` attribute and an `encoding` attribute.

As with the `wctp-Alphanumeric` element, the payload data is specified as PCDATA and since the data cannot be explicitly represented using XML, the data must be encoded. The encoding scheme may be specified with the `encoding` attribute as either **standard** or **base64** encoding.

The default mechanism for encoding the data is **base64** as defined in RFC 1341, Section 5.2 Base64 Content-Transfer-Encoding. Alternatively, XML provides a **standard** canonical form that can be used to specify numeric, alphanumeric, and some 8-bit data. The 8-bit binary data can be represented as an ASCII string that encodes a hexadecimal value using a hexadecimal character reference (e.g. &#xD1; represents the binary octet 11010001), however, not all octets can be encoded in XML. In particular, most of the “special characters” with hexadecimal values 0x00 to 0x1F cannot be encoded in XML and are considered illegal characters. Because of these limitations, base64 encoding should usually be used. The following two examples show how binary data may be encoded.

```
Binary Data:

00000000 504B 4334 2AA0 BBDE FF7F 6A90 6F2C 9539 PKC4*.....j.o,.9

00000010 854B DC64 FF48 2C50 66FF 2B33 3942 5361 .K.d.H,Pf.+39BSa

Standard XML Encoded Data:

PKC4*&#xa0;&#xbb;&#xff;&#x7f;j&#x90;o,&#x95;9;&#x85;K;&#xdc;d&#xff;
H,Pf&#xff;+39BSa
```

Example 1 Standard XML Encoding

```
Binary Data:
00000000 504B 4334 2AA0 BBDE FF7F 6A90 6F2C 9539 PKC4*.....j.o,.9
00000010 854B DC64 FF48 2C50 66FF 2B33 3942 5361 .K.d.H,Pf.+39BSa


Base64 Encoded Data:
UEtDNCqgu97/f2qQbyyVOYVL3GT/SCxQZv8rMzlCU2E=
```

Example 2 Base64 Encoding

Two mechanisms are allowed for the transfer of transparent data over the air. The `type` attribute is used to select either **OPAQUE** or **FLEXsuite**. Opaque is the default type implying general binary data. Data that follows Motorola FLEXsuite defined form should have a type defined as **FLEXsuite**. Note that **FLEXSuite** is also accepted in order to provide backwards compatibility with earlier versions of WCTP.

7.3 Confirmation Response

The Confirmation operation of WCTP (`wctp-Confirmation`) is used to report the success or failure of the submission of a request for processing. Figure 8 - WCTP Confirmation Operation shows the major XML elements of the Confirmation operation. Each of these elements specifies a number of different attributes or parameters associated with it.

The Confirmation is returned for each HTTP POST, and conveys positive or negative acknowledgement that the message was received for delivery.

Structurally, the Confirmation operation contains either the Success or the Failure element. Note: `WCTP-Success` is not an indication that the message has been delivered; rather it indicates the gateway has accepted the message for delivery.

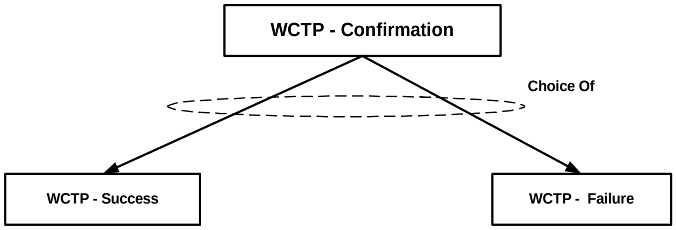

Figure 8 - WCTP Confirmation Operation

7.3.1 Confirmation Success

A `wctp-Success` element must be returned as part of any `wctp-Confirmation` that is an indication of a successful operation. It is not an indication to guarantee delivery of a message, nor any guarantee of a particular type of response. Rather it is the acknowledgement of any operation being accepted and understood as a valid WCTP operation. In addition the `successCode` and `successText` will give indication as to the type or condition of the successful outcome of the previous operation. Appendix E Error and Success Codes describes any valid values for these attributes.

7.3.2 Confirmation Failure

A `wctp-Failure` element must be returned as part of any `wctp-Confirmation` that is an indication of a failed operation. There are many explicit types of `errorCode` and `errorText` combinations defined in Appendix E Error and Success Codes. There is also a range of error codes that are experimental or carrier gateway specific.

7.4 Response Header

WCTP provides response headers for Enterprise Hosts, Polling Enterprises, and Transient Clients. In many cases these attributes have identical meaning for all the actors, but there are additional specific attributes for some of the actors as defined in Sections 8 Enterprise Hosts and 9 Transient Client.

These common attributes that may provide additional meaningful information for specific types of applications are `responseTimestamp` and `respondingToTimestamp`. The `responseTimestamp` should coincide with the time that the wireless network received the response from the wireless device. The `respondingToTimestamp` should mirror the `submitTimestamp` in the original message that was submitted to the wireless carrier.

7.5 Notification

The Notification response (`wctp-Notification`) may indicate a message delivery status condition such as "message accepted by the wireless device", "message has been delivered to the wireless device" or "message read or processed by the wireless device." The types of notification status responses desired must be specified when the message is created. The values for the `type` attribute are “**QUEUED**”, “**DELIVERED**”, or “**READ**”.

If a gateway does not detect an error on message entry but an error is detected later, a Failure response (`wctp-Failure`) may be delivered asynchronously using the `wctp-StatusInfo` to Enterprise Hosts or Polling Enterprises. A Transient client would receive a notification failure via the `wctp-ClientStatusInfo` Operation. This Failure response may indicate a permanent or temporary failure. The specifics of the failure condition are reported via a standardized `errorCode` and optional `errorText`.

8 Enterprise Hosts

An enterprise may choose to implement WCTP either as an enterprise host or as a polling enterprise. Each has the ability to submit messages and request information that is not available to transient clients (as defined in section 9 Transient Client). The only recommended reason for implementing a polling enterprise is due to the absence of a static connection to the Internet. Any security concerns can be resolved by implementing a proxy server (as defined in section 3.3.5 Proxy Server).

8.1 Pushing WCTP from the Wire line to Wireless Network

A wire line system sends (or pushes) information to the carrier gateway using the HTTP protocol. This is the same protocol used by browser software to request and receive information from the World Wide Web and to forward information to Internet based servers to perform some function based upon the data forwarded (such as a search function, a database lookup, adding a name to a mailing list or ordering goods).

HTTP is a client/server protocol. An HTTP client always makes a request to an HTTP server. HTTP supports two *major* processes. Clients may GET (or pull) information from a server (such as a Web page) or POST data to a server. Regardless of which process is used, there is always a response sent to the request. The response always contains an HTTP protocol header field optionally followed by some returned data.

In WCTP, the HTTP POST method is used to send a WCTP formatted request (WCTP Operation) to the WCTP server. The response by the server is a WCTP XML formatted response (another WCTP control block or Operation).

The HTTP content type to be used in WCTP communications is: `text/xml`. The following diagram shows the normal call flow from the client to the server when a wire line client sends a request to the wireless network.

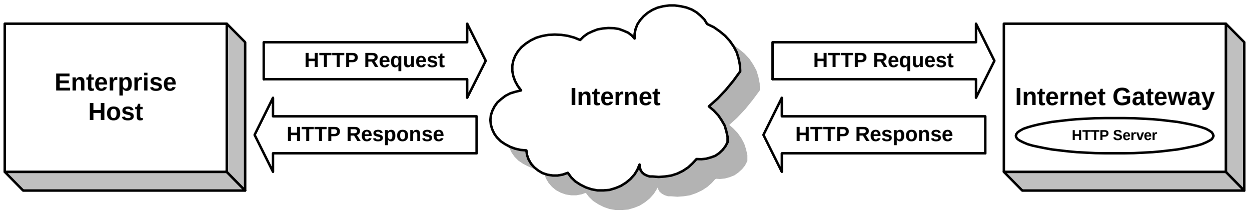

Figure 9 - Pushing WCTP Operations to the Wireless Network

8.2 Send Message

There are three types of message submission operations for enterprises. They are `wctp-SubmitRequest`, `wctp-SendMsgMulti` and `wctp-MessageReply`.

In all cases, servers may impose a maximum submission rate on messages coming into the wireless network. If this rate is exceeded, an error response code is returned in lieu of accepting the message to indicate that the allowed input rate has been exceeded. If this rate is exceeded an excessive number of times, another error response code may be returned indicating that the sending system is subject to being disabled from sending further messages.

8.2.1 WCTP Submit Request Operation

The "Submit Request" operation of WCTP (`wctp-SubmitRequest`) is used to initiate a new message from an enterprise host, polling enterprise or carrier gateway. Figure 10 shows the major XML formatted elements of the `wctp-SubmitRequest`. Each of these elements may specify a number of different attributes or parameters that are associated with the element.

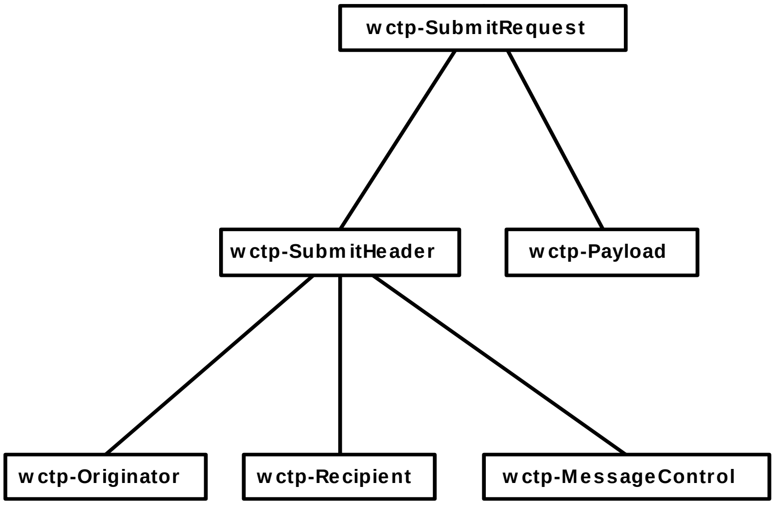

Figure 10 - WCTP Submit Request Operation

The `wctp-SubmitRequest` has two major elements, the `wctp-SubmitHeader` and the `wctp-Payload`.

The response to the `wctp-SubmitRequest` operation is a `wctp-Confirmation.`

8.2.1.1 WCTP Submit Header

The `wctp-SubmitHeader` provides information as to the message originator, message recipient, and many message control parameters.

Message Control (`wctp-MessageControl`) parameters allow the message sender to control when a message is to be delivered and what types of status notifications should be reported. It is also possible to direct responses to a different Enterprise host or application than the sender of the original message.

One of the key parameters is the `messageID`. This is a unique message identifier assigned by the message originator. This identifier will be contained in any subsequent responses returned on behalf of the submitted message.

A transaction identifier (`transactionID`) may be submitted with the message. If such an ID is included, it will be returned in subsequent responses. The originator of a message uses a `transactionID` to tag a group of individually submitted requests. These requests might even be sent to a number of different devices that may even span several different wireless networks. The `transactionID` may be used to indicate that all of these requests where part of a particular instance of an application on behalf of a sequence of operations that the application may have been performing.

8.2.2 Send Multiple Recipient Message Operation

The send multiple recipient message operation of WCTP (`wctp-SendMsgMulti`) is used to initiate a new multiple-recipient message from a wire line Enterprise Host. Figure 11 shows the major XML formatted elements of the Send Multiple Recipient Msg operation. Each of these elements may specify a number of different attributes or parameters associated with the element.

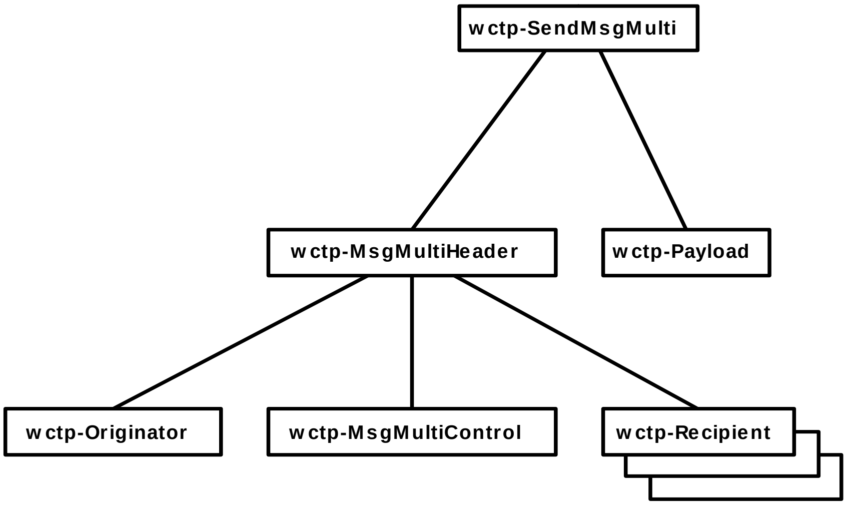

Figure 11 - WCTP Send Multiple Recipient Msg Operation

The send multiple recipient message operation has two major elements, the `wctp-MsgMultiHeader` and the payload. The response to a `wctp-SendMsgMulti` is a `wctp-SendMsgMultiResponse`.

In the case of the `wctp-SendMsgMulti`, servers may impose a maximum number of recipients on messages coming into the wireless network. If this rate is exceeded, an error response code is returned in lieu of accepting the message to indicate that the limit has been exceeded.

8.2.2.1 Msg Multi Header

The Msg Multi Header provides information about the message originator and message recipients, and establishes many message control parameters. The structure of the `wctp-MsgMultiHeader` element is identical to the Submit Header from the Submit Request operation with two exceptions: it supports multiple recipients, and includes an `allRecipsRequired` attribute.

8.2.2.2 Multiple Recipients Details

In some cases, Wire line Enterprise Hosts will submit a message where one or more of the recipients are invalid, but at least one is valid. In this case, the Wire line Enterprise Host controls whether the message is sent to the valid recipients or not by setting the allRecipsRequired flag in the wctp-MsgMultiControl element. If the user sets allRecipsRequired to *true* and one or more of the recipients are invalid, then the message will not be sent to any of the recipients. If the user sets allRecipsRequired to *false*, then the message will be sent to all valid recipients, regardless of how many of the recipients are invalid.

8.2.2.3 Limit Details

Wireless Systems shall set and enforce a maximum number of recipients that they will accept on a single message. Wireless Systems shall accept and validate all recipients on a message as long the messages contain no more than this maximum number of recipients. Wireless Systems shall reject all messages submitted with more than this maximum number of recipients.

If a Wireless System supports at least two recipients on a wctp-SendMsgMulti, it supports “multiple recipients”.

8.2.2.4 Multiple Recipient Response

The Send Multiple Recipient Msg Response operation *(*wctp-SendMsgMultiResponse) is sent by a WCTP server of a wireless carrier to a wire line system in a response to a Send Msg Multi operation. Each Send Msg Multi Response operation may contain a batch of wctp-FailedRecipient elements to tell the wire line system that one or more of its recipients is invalid (unacceptable to the carrier).

There are three attributes dealing with the quantities of recipients that may be submitted or have been processed. They are maxNumRecips, numValidRecips, and numInvalidRecips. The maximum number of recipients that a carrier gateway will accept for deliver is expressed as maxNumRecips. The synchronous response to wctp-SendMsgMulti will inform the sender of the number of valid recipients via the numValidRecips attribute and the number of invalid recipients via the numInvalidRecips attribute.

The WCTP gateway should attempt to validate the list of recipients as soon as possible in order to generate meaningful response information in the wctp-SendMsgMultiResponse. When this is not possible, the WCTP gateway will generate wctp-StatusInfo elements containing wctp-Failure elements as necessary to notify the sender of invalid recipients.

8.2.2.5 wctp-failedRecipient

For each recipient that a WCTP gateway deems invalid, a wctp-FailedRecipient element is included in the wctp-SendMsgMultiResponse. The element contains a recipientID to indicate which recipient failed and an errorCode and errorText to indicate why delivery to the recipient failed.

8.2.3 Message Reply Operation

The Message Reply operation of WCTP (wctp-MessageReply) is used to report message replies returned on behalf of other messages. It may be a reply to a message submitted wirelessly or over wire line including those submitted through the operations wctp-SubmitRequest, wctp-SendMsgMulti, and wctp-MessageReply WCTP provides threading information to tie Message Replies to the original message. More than one reply may be returned for each message submitted. Figure 12 - WCTP Message Reply Operation shows the major XML elements of the Message Reply operation. Each of these elements may specify a number of different attributes or parameters, which are associated with the element. The Message Reply contains two main elements: the `wctp-Response-Header` and the `wctp-Payload`. Specification of the message payload follows the same rules as listed in Section 7.2 Payload as headers follows those listed in Section 7.1 Header.

8.2.3.1 Response Header

The Response Header provides information regarding the message originator, the message recipient, and control information to further specify how the Message Reply should be processed. The Originator (`wctp-Originator`) portion of the Response Header indicates the author of the reply. In the case of a response from a wireless device, this would specify the unique address of the device as specified by the parent wireless carrier.

The Recipient (`wctp-Recipient`) information of the Response Header would normally contain the address of the sender of the original message. If, when the message was submitted, it was indicated that responses should be sent to a different destination other than the message originator, this destination information will appear in the `wctp-Recipient` instead.

The Message Control element of the Response Header contains the unique Message Identifier (`responseToMessageID`) that ties the response back to a particular submitted message. The structure of the Message Control element is identical to its namesake from the Submit Header of the Submit Message Request operation (see Section 8.2.1.1 WCTP Submit Header).

There are additional optional attributes that may provide additional meaningful information for specific types of applications. These are `responseTimestamp, respondingToTimestamp, and onBehalfOfRecipientID`. The `responseTimestamp` and `respondingToTimestamp` are defined in section 7.4 Response Header. The `onBehalfOfRecipientID` attribute may be used in special circumstances when a wireless network sends a message informing an Enterprise Host, or Polling Enterprise of a significant event regarding a previous message or the wireless network itself.

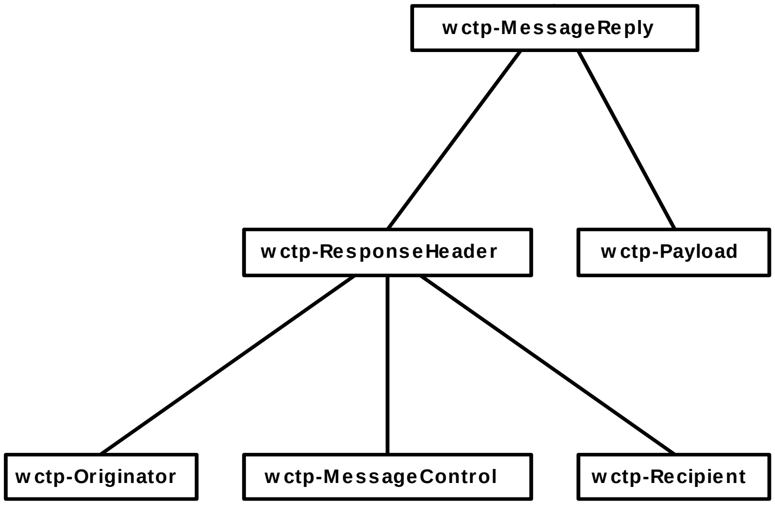

Figure 12 - WCTP Message Reply Operation

8.3 Subscriber Lookup

Enterprise Hosts may use the Lookup Subscriber operation (`wctp-LookupSubscriber`) to query the capabilities of a wireless device. Wireless carriers are not required to support this operation.

The synchronous response to the Lookup Subscriber operation is a WCTP Confirmation. Asynchronously the Lookup Subscriber Response operation (`wctp-LookupResponse`) is sent by a WCTP gateway. The response to a Lookup Subscriber specifies a set of attributes describing capabilities of the device. Such information can be used to appropriately format any subsequent messages. Typically, an Enterprise host uses a Lookup Subscriber operation to obtain information about a specific subscriber’s messaging device that may be useful when interacting with the device.

The Lookup Subscriber Operation contains three elements: the `wctp-Originator`, `wctp-LookupMessageControl` and `wctp-Recipient`. The `wctp-Originator` and `wctp-Recipient` attributes are as described in section 7 Common Elements. The `wctp-Recipient` indicates the subscriber device whose properties are being queried with the Lookup Subscriber operation.

8.3.1 wctp-LookupMessageControl

The `wctp-LookupMessage` Control element contains a `messageID` and `transactionID` (both are defined in section 8.2.1.1 WCTP Submit Header) that may be used by an Enterprise Host to correlate the request with a subsequent response in a way identical to the usage of these attributes as described in section 7 Common Elements. The `sendresponsesToID` is used to redirect the response the Lookup Subscriber operation to an entity other the originator as indicated in `wctp-Originator`.

8.4 Subscriber Device Location Lookup

This operation behaves in a similar manner to the Subscriber Lookup Operation. Enterprise Hosts may use the Device Location operation (`wctp-DeviceLocation`) to determine the location of a wireless device. Wireless carriers are not required to support this operation. If supported, this operation can return either an approximate location determined by the network, or a more precise location which may be determined by the device itself. The approximate location may be the location of the last base station that the subscriber device was in contact with.

The synchronous response to the Device Location operation is a WCTP Confirmation. Asynchronously the Device Location Response operation (`wctp-DeviceLocationResponse`) is sent by a WCTP gateway. The response to a Device Location operation specifies a set of attributes describing the location of the device.

The Subscriber Device Location Operation contains three elements: the `wctp-Originator, wctp-DeviceLocationControl` and `wctp-Recipient`. The `wctp-Originator` and `wctp-Recipient` attributes are as described in section 7 Common Elements. The `wctp-Recipient` indicates the subscriber device whose location is being queried with the Device Location operation.

8.4.1 wctp-DeviceLocationControl

The `wctp-DeviceLocationControl` element contains a `messageID, transactionID, sendResponseToId and a fixType. The messageID and transactionID` (both are defined in section 8.2.1.1 WCTP Submit Header) may be used by an Enterprise Host to correlate the request with a subsequent response in a way identical to the usage of these attributes as described in section 7 Common Elements. The fixType fields in the request and reply provide information regarding the desired and actual quality of location data. While the actual network behavior is implementation-dependant, the recommended levels of location fix quality are:

- *Approximate*: a crude location fix, possibly the location of the nearest receiver to the device.
- *Estimated*: a more refined fix, possibly involving triangulation from multiple receivers.
- *Precise*: the highest quality of location fix supported by the network, possibly involving communication with the device to establish its location.

The `fixType` in the lookup request specifies the maximum desired quality of the location fix, and defaults to *estimated.* The network will not initiate a *precise* location fix except under explicit instructions, since this may involve over-the-air network traffic. In all cases, the response specifies the actual achieved quality of the location fix. For example, if the default of *Estimated* is specified, the system may return *Approximate* if that is what is available.

The sendResponseToID in the request specifies that the response is to be redirected to an entity other than the originator as indicated in wctp-Originator. One possible use for this is to provide better security by not using the senderID as both a publicly visible wctp address and a user identifier. If authentication is to be used (as described below) then there must be a user identifier to go with the authentication password provided. If sendResponseToID is used to identify the wctp address to send the response, then the senderID does not need to be a wctp address, and does not need to be exposed.

If authentication is required then the `securityCode` attribute of the `wctp-Originator` element or the `authorizationCode` attribute of the `wctp-Recipient` element should be used. Use of these elements will be implementation dependant. The host may limit access to certain subscriber's location information to a specific wctp-Originator. In this case the host must implement a method to keep track of the mapping of wctp-Originator to subscriber range along with the password used by the wctp-Originator for authentication.

Use of the authorizationCode in the wctp-Recipient can be context dependant. This means that a separate code can be used for different subscriber features. For example, one code can be used to control the ability to send messages to a particular subscriber and a different code can be used to control access to the subscriber's location information.

8.5 Asynchronous Operation

Even though wctp is a request / response protocol, there are times when the request cannot be immediately fulfilled. The most obvious example of this delay is a delivery notification. A user sends a message using `wctp-SubmitClient` and requests a delivery notification. But the notification is not available until the message is delivered. Therefore, the notification must be delivered asynchronously of the original request. Since looking up subscriber information may be a slow process, the Lookup Subscriber Response and the Device Location Response are also asynchronous operations.

8.5.1 WCTP Status Info Operation

The Status Info operation of WCTP (`wctp-StatusInfo`) is used to report both message delivery failures as well as notifications returned on behalf of other messages. Figure 13 shows the major XML elements of the Status Info operation. Each of these elements specifies a number of different attributes or parameters. The Status Info contains two main elements: the Response Header and either a Failure (section 7.3.2) or a Notification (section 7.5).

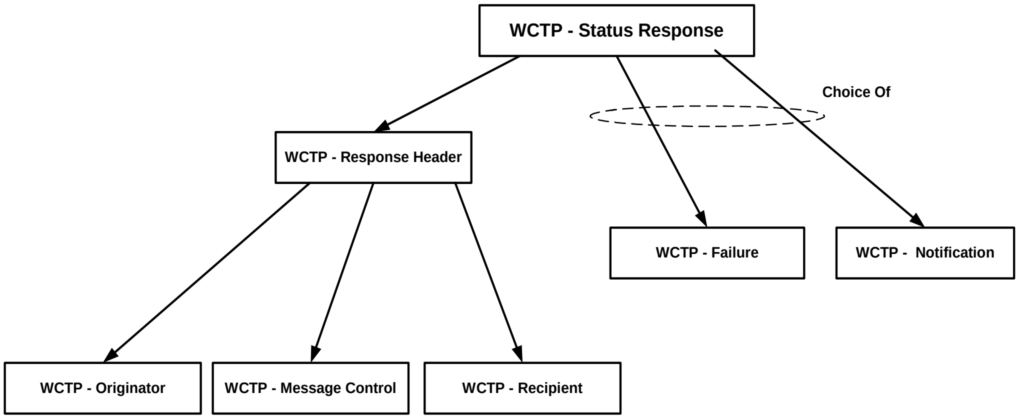

Figure 13 - WCTP Status Response Operation

The Response Header provides information with regard to the message originator (the recipient of the original message), the message recipient (the sender of the original message), fields that further identify the original message (`messageID`, `transactionID`, etc), and control information to further specify how the response should be processed.

8.5.2 Lookup Subscriber Response Operation

The Lookup Subscriber Response operation (`wctp-LookupResponse`) is sent from a wireless carrier to an enterprise host in a response to the Lookup Subscriber operation. The synchronous response to this operation is a `wctp-Confirmation`.

The Lookup Subscriber Response Operation contains the following elements: the `wctp-Originator, wctp-Recipient, and wctp-LookupData or wctp-Failure`. The `wctp-Originator`, `wctp-Recipient`, `wctp-Failure` attributes are as described in the Common elements section. The `wctp-Originator` indicates the subscriber device whose properties are being queried with the Lookup Subscriber operation.

The `responseToMessageID` should match the `messageID` in the Lookup Subscriber Operation. The Response Operation may contain a `transactionID` (as defined in section 8.2.1.1 WCTP Submit Header).

8.5.2.1 wctp-LookupData

The attributes in `wctp-LookupData` specify important characteristics of a subscriber device. The attribute `maxMessageLength` indicates the maximum number of characters that may be sent in a single message to the device. The `mcrSupported` attribute indicates whether or not the device supports MCR messages. The `canRespond` attribute indicates whether or not the device is capable of generating replies to messages sent to the device.

8.5.3 Device Location Response Operation

The Device Location Response operation (`wctp-DeviceLocationResponse`) is sent from a wireless carrier to an enterprise host in a response to the Device Location operation. The synchronous response to this operation is a `wctp-Confirmation`.

The Device Location Response Operation contains the following elements: the `wctp-Orinator, wctp-Recipient, and wctp-LocationData` or `wctp-Failure`. The `wctp-Originator`, `wctp-Recipient`, `wctp-Failure` attributes are as described in the Common elements section. The `wctp-Originator` indicates the subscriber device whose location is being queried with the Device Location operation.

The `responseToMessageID` should match the `messageID` in the Device Location Operation. The Device Location Response Operation may contain a `transactionID` (as defined in section 8.2.1.1 WCTP Submit Header).

8.5.3.1 wctp-LocationData

The attributes in `wctp-LocationData` specify the location of a subscriber device. The attribute `status` indicates the quality of the data, and exactly how much information the network infrastructure is able to report about the device. The values can be as follows:

- ok – The fix is valid.
- `invalid` – No information is available, or some network error occurred. In this case, any value returned for `fixType` is also not valid.
- `noalt` – No altitude information is available.
- `nolonglat` – No longitude or latitude information is available.
- `stale` – The fix information provided is out of date. The time period for determining this is implementation and application dependant and is not defined here.

The attribute `fixType` indicates the quality of fix that is being returned. Note that the system should always return the best possible fix that is available, so the value returned for `fixType` may not match the `fixType` requested. The values for `fixType` are described in 8.4.1wctp-DeviceLocationControl. The attribute `fixTimeStamp` indicates the time when the fix was made. The attributes `longitude`, `latitude` and `altitude` indicate the location of the device. `longitude` and `latitude` are specified in a format similar to the GPS NMEA format. This format is described in section 9.8. For example, 123 degrees 15.0635 minutes West longitude would be 12315.0635W. `altitude` is meters above or below sea level.

8.6 Message Delivery to a Wire line Network

The wireless network can forward data to the wire line network in the HTTP response to a previous HTTP request as shown in Figure 5 - Call Flow Between Network Components. But when the wireless network has unsolicited information available, it would like to push this data out to the wire line system. There are several methodologies employed within the Internet today to "push" information from servers to client machines. Two forms of push technology are currently recommended by this specification.

8.6.1 Non Polling Enterprise Operations

The recommended means of pushing information from the wireless network to the wire line system is also through the use of the HTTP protocol. In this case, the TCP/IP connection is initiated by the wireless network and an HTTP request carries a POST operation that contains an XML formatted WCTP Operation as its data content. The WCTP Operation carried in the HTTP response from the wire line system indicates if the data sent has been accepted, processed and/or safe-stored by the wire line system. If so, the wireless network can drop the WCTP Operation from its queues knowing that the information has been properly conveyed to the wire line system.

The major requirement to support this type of push technology is that the wire line system must be running some type of HTTP-based web server-like software in order to accept and process the WCTP Operations pushed from the wireless network. A minimum requirement is to accept HTTP 1.0 posts in accordance with RFC 1945.

In support of this form of push, addressing information carried by the WCTP protocol provides the host and domain name of the wire line server to process the pushed data. The standard WCTP message routing mechanism derives the host and domain name from the domain portion of the recipient ID.

8.6.1.1 Return To Service Operation

The Return To Service operation of WCTP (`wctp-ReturnToSvc`) is used by an enterprise host's application to inform a carrier's gateway that the application is alive and waiting for messages to be POSTed to it. The response to the return to service operation is a `wctp-Confirmation`. Each application that returns to service is to issue its own return to service operation. The response to a wctp-ReturnToSvc is a wctp-Confirmation.

The Return to Service provides the address of the application that is listening for POSTed messages. It may be expressed as a WCTP address or as a WCTP alias and must conform to all of the rules and defaults for those types.

The Return To Service operation may also be used when an application has noticed that it has not received messages for an unusually long period of time. However, developers must be judicious in this use. Carriers will monitor Return To Service operations to detect denial of service attacks. Offending IP addresses may be blocked at the firewall or other actions may be taken against the attacker.

8.6.2 Polling Enterprise Operations

If an Enterprise Host's network is not able to run an HTTP server to accept pushed data (for security reasons or because of the lack of an HTTP server) from the wireless network, a form of pseudo-pushing may be used. When this form is used, the wireless network queues the data that normally would have been pushed. Then the wire line system initiates delivery of the accumulated messages by issuing a Poll For Messages operation.

The response to a Poll For Messages operation, the Poll Response operation, may return a batch of messages, each identified by an assigned sequence number.

When a wireless gateway receives a message and encounters a `pollerID` in the domain portion of the `recipientID`, the wireless network does not attempt to push the data to the Enterprise Host. Instead, it holds the data in its queues and waits for the Enterprise Host to submit a `wctp-PollForMessages` operation.

For a wire line system to support the `wctp-PollForMessages` operation, it must be registered at the wireless network and have a unique poller ID. When sending a message to a wireless device, the wire line system uses the unique `pollerID` as part (the domain portion) of the `senderID` (or the send response to ID). A wireless device may send unsolicited messages to the wire line system by using the `pollerID` as part (the domain portion) of the `recipientID`.

8.6.2.1 Polling For Messages

The `wctp-PollForMessages` operation is used by a Polling Enterprise to request and potentially receive messages (`wctp-StatusInfo`, `wctp-MessageReply`, `wctp-SubmitRequest`, and `wctp-LookupResponse`) via the poll response (`wctp-PollResponse`). These responses may be notifications or replies for messages which were previously submitted, or they may be unsolicited messages to be processed by the Enterprise host.

When submitting a `wctp-PollForMessages` operation, the Enterprise Host must supply its unique `pollerID` and password (`securityCode`). Note: the `pollerID` and `password` combination must be registered with the carrier.

In order to speed up the transfer of messages queued at the wireless network; a Poll Response operation (`wctp-PollResponse`) may return multiple messages (status responses, replies, delivery notifications or unsolicited messages). The maximum number of messages the Enterprise Host can accept in one transaction may also be passed in the Poll For Messages Operation as the `maxMessagesInBatch` parameter. (Note that the system responding to the polling operation may also limit the maximum number it is willing to deliver at once. Therefore, it is not guaranteed that the `maxMessagesInBatch` messages will be returned even if they are queued. This detail is carrier implementation specific.)

Figure 14 shows the major elements of the Poll For Messages Operation.

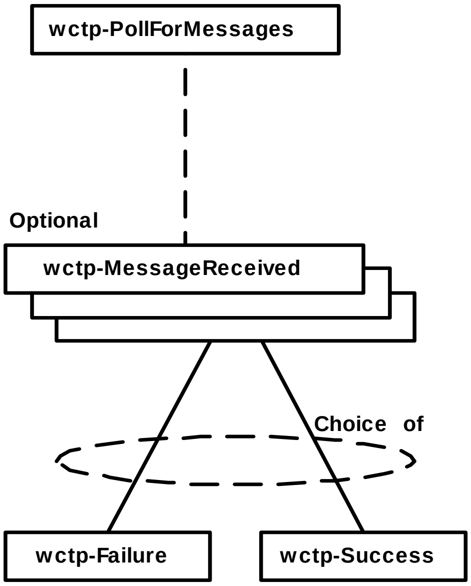

Figure 14 – WCTP Poll for Messages Operation

8.6.2.1.1 MessageReceived

The Enterprise Host must acknowledge all WCTP Operations delivered so that the wireless system may remove them from its queues. The Message Received section of the Poll For Messages Operation is used to acknowledge these deliveries. Multiple Message Received elements, each containing a `sequenceNo`, may be specified in this section so that multiple messages may be acknowledged at once. If this acknowledgement is not returned with the subsequent poll, the non-confirmed messages will be delivered again.

Note: The wireless provider may meter the load on its system by returning a `minNextPollInterval` in the Polling Response. That is the number of seconds the Enterprise Host is to delay before polling for more messages. However, an Enterprise Host may want to acknowledge message delivery immediately (before the delay interval has expired). This is allowed by setting the `maxMessagesInBatch` field to `0`. However, when requesting the delivery of more messages (maxMessagesInBatch > 0), the delay returned on the last actual request for messages must be observed.

8.6.2.2 Polling Response

A `wctp-PollResponse` is generated in response to a `wctp-PollForMessages`. It may indicate that there is no data to return to the wire line system or it may return one or more queued WCTP Operations.

A sequence number (`sequenceNo`) is assigned to each `wctp-Message` delivered in a `wctp-PollResponse`. Each sequence number must be returned in a `wctp-MessageReceived` element in a subsequent Poll operation to acknowledge its successful delivery. If a sequence number is not acknowledged in the next `wctp-PollForMessages`, the message will be delivered again. Note that the sequence numbers need only be unique, not necessarily in sequence.

The wireless network may impose implementation-specific maximum polling rates passed in the “`minNextPollInterval`” field. The value specified by this attribute is the minimum number of seconds that an enterprise host must wait before polling again. This is intended to allow a carrier the ability to prevent excessive traffic and work load resulting from repetitive polling. The polling rate may vary based time of day, system load, or other conditions the carrier feels significant. System error response codes are returned in the `wctp-PollResponse` to indicate when a polling rate has been exceeded or when polling rates have been excessively exceeded. If excessive polling continues, the wireless network may permanently disable the offending `pollerID`. If an enterprise application does not require a polling interval as small as the one provided by the carrier, then it should use its own longer interval as a “default” polling interval.

Figure 15 shows the major elements of the Poll Response Operation.

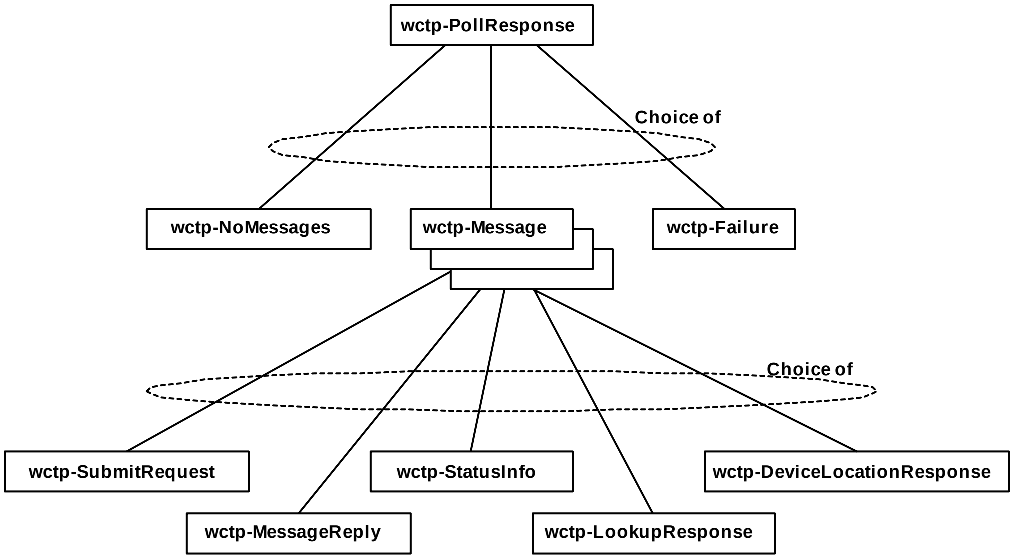

Figure 15 – WCTP Poll Response Operation

8.6.2.2.1 wctp-Message

When there are no outstanding messages queued at the wireless network, a `wctp-NoMessages` is returned inside of the `wctp-PollResponse`.

If there is an error such as polling interval exceeded or an invalid poller ID, a wctp-Failure is returned inside of the `wctp-PollResponse`.

However the wctp-Message is the most common content returned inside of the `wctp-PollResponse`. Several wctp-Messages may be returned inside one `wctp-PollResponse` (depending on the carriers limitation, the Enterprise Host's limitations expressed in `maxMessagesInBatch`, and the number of messages queued). The wctp-Message may be a `wctp-StatusInfo` (containing a notification message providing the status of a previously sent message), `wctp-MessageReply` (containing a reply to a previously sent message), `wctp-SubmitRequest` (containing a new message from a wireless device), `wctp-LookupResponse` (containing the results of a previous subscriber lookup), or wctp-DeviceLocationResponse (containing the results of a previous location query).

9 Transient Client

The role of a transient client is defined in section 3.3.4 Transient Client.

9.1 Identifiers For Transient Clients

The format of identifiers for Transient clients in WCTP is not fixed. Normally, uniqueness of such an identifier cannot be guaranteed. Therefore, wireless systems cannot rely on the sender identifier for messages submitted by Transient clients. In case of Transient clients, for delivery of responses the wireless system must rely on the uniqueness of an assigned tracking number or a combination of tracking number and recipient identifier of the original message. The senderID, specified on a submitted message by a Transient client, must be specified for authentication when querying for status.

9.2 WCTP Submit Client Message Operation

The Submit Client Message operation of WCTP (`wctp-SubmitClientMessage`) is used to initiate a new message from a Transient client to the WCTP Carrier Server. Figure 16 shows the major XML elements of the Submit Client Message operation. Each of these elements specifies a number of different attributes or parameters associated with it.

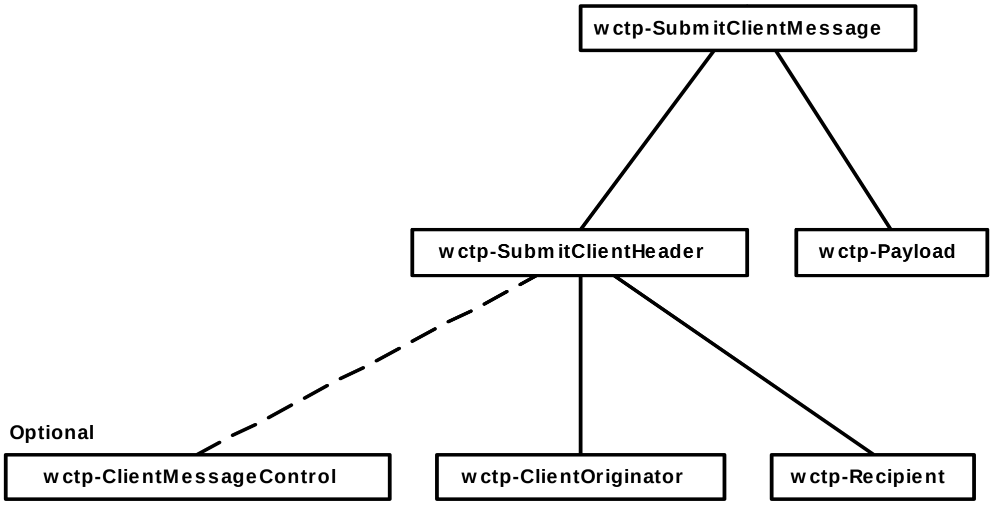

Figure 16- WCTP Submit Client Operation

The Submit Client Message has two major elements: `wctp-SubmitClientHeader` and `wctp-Payload`.

9.2.1 wctp-SubmitClientHeader

The `wctp-SubmitClientElement` provides information about the message originator and message recipient and establishes many message control parameters. The attribute `submitTimestamp` may optionally be set to the time the message was submitted. No behavior is implied by the `submitTimestamp` attribute.

9.2.1.1 wctp-ClientOriginator

The `wctp-ClientOriginator` element contains information describing the originator of the message. The `senderID` may be set to any value useful for describing the sender. Unlike the `senderID` attribute of `wctp-Originator`, it is not interpreted as a WCTP address. Some servers, however, may require the `senderID` in combination with the `trackingNumber` and `recipientID` to uniquely identify a message.

The `miscInfo` field is an optional parameter with no implied behavior.

9.2.1.2 wctp-ClientMessageControl

The `wctp-ClientMessage` control element contains attributes that establish how the message will be delivered and what type of notifications will be issued when submitting a message.

9.2.2 Payload

The structure of the Payload element is described in the Section 7.2 Payload.

9.2.3 Response

The Submit Client Response operation (`wctp-SubmitClientResponse`) is sent from a WCTP server of a wireless carrier to a Transient client in a response to the Submit Client Message operation.

The Submit Client Response operation provides positive or negative acknowledgement of the submission of the message. In the case of positive acknowledgement the WCTP server provides the message sender with a tracking number assigned to the message. This tracking number, in combination with the `senderID` and the `recipientID` of the message, may be subsequently used to check for replies and status notifications through a Client Query Operation (See Section 9.3 WCTP Client Query Operation). There are additional optional attributes that may provide additional meaningful information for specific types of applications. They are `responseTimestamp, respondingToTimestamp` as defined in section 7.4 Response Header.

The response to a Submit Client Message operation of WCTP (`wctp-SubmitClientMessage`) is a Submit Client Response operation (`wctp-SubmitClientResponse`*)*. Figure 17 shows the major XML elements of the Submit Client Response operation. Each of these elements specifies a number of different attributes or parameters associated with it.

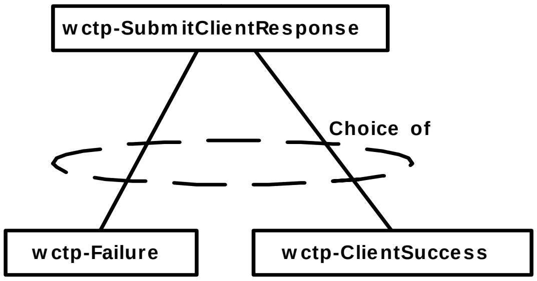

Figure 17 - WCTP Submit Client Response Operation

The Submit Client Response has two major elements: `wctp-ClientSucess` and `wctp-Failure`.

9.2.3.1 wctp-ClientSuccess

The `wctp-ClientSuccess` element is identical that of the `wctp-Success` element as defined in section 7.3.1 with the exception of the `trackingNumber` attribute. The `trackingNumber` is published by the carrier gateway to allow a transient client to use the Client Query operation to obtain the status of the submitted message. The format of the `trackingNumber` is not specified by the WCTP protocol and is implementation specific.

9.2.3.2 wctp-Failure

The structure and behavior of `wctp-Failure` is defined in section 7.3.2.

9.3 WCTP Client Query Operation

A Client Query operation (`wctp-ClientQuery`) is a form of a poll operation used by Transient clients to check for status information or responses to a previously submitted message. The response to a Client Query operation (`wctp-ClientQueryResponse`) can indicate that no information is available, or it may return a collection of replies (`wctp-ClientMessageReply`) and message statuses (`wctp-ClientStatusInfo`).

To perform the Client Query operation the `senderID`, the `recipientID` and the tracking number of the original message must be supplied. The sender ID field in a transient operation must not be used as a routing or addressing component. It is only used as a key to ensure that any status or reply information is delivered to the correct entity.

The Client Query Response operation (`wctp-ClientQueryResponse`) is sent to a WCTP server of a wireless carrier to a Transient client in response to a Client Query operation. **Figure 18** shows the major XML elements of the Client Query Response operation.

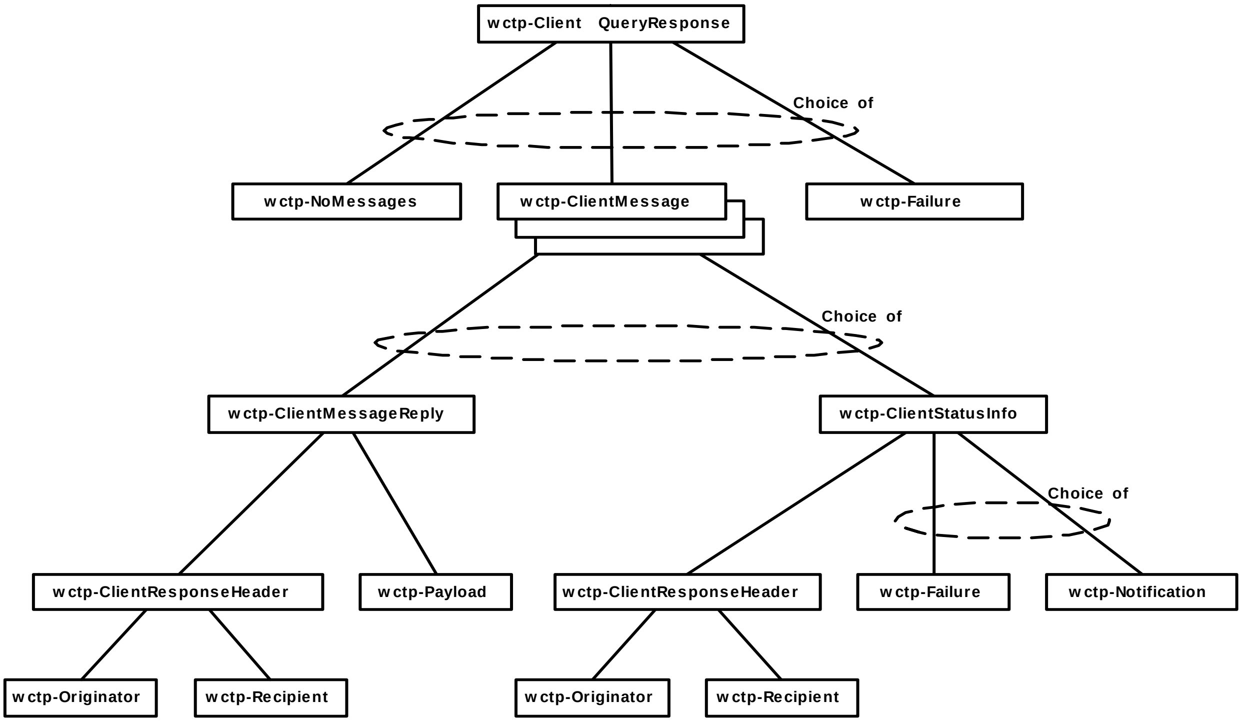

Figure 18 - WCTP Client Query Response Operation

The Client Query Response operation provides the requestor with a set of Client Messages. Each Client Message contains either a Message Reply or a Status Info to the message identified by a combination of the `senderID`, the `recipientID` and the tracking number as specified in the Client Query request.

Alternatively, the No Messages element is used to indicate to the requestor that there are no replies or status information for the original message.

Note: The carrier gateway may meter the load on its system by returning a `minNextPollInterval` in the `wctp-ClientQueryResponse`. That is the number of seconds the Transient Client is to delay before attempting another `wctp-ClientQuery` with identical values for `senderID`, `recipientID`, and `trackingNumber`.

Each time the Client Query operation is submitted, the Client Query Response operation will return a set of replies and status notifications with the respect to the original message. WCTP servers will hold this information for an implementation-specific period of time.

Appendix A WCTP Document Type Definition

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
<!-- file-url='http://dtd.wctp.org/wctp-dtd-v1r3.dtd' -->
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->

<!-- Wireless Communication Transfer Protocol (WCTP)                  -->

<!-- ============================= -->
<!-- Declaration of wctp-Operation -->
<!-- ============================= -->

<!ELEMENT wctp-Operation            ( wctp-ClientQuery
                                    | wctp-ClientQueryResponse
                                    | wctp-Confirmation
                                    | wctp-LookupSubscriber
                                    | wctp-LookupResponse
                                    | wctp-DeviceLocation
                                    | wctp-DeviceLocationResponse
                                    | wctp-MessageReply
                                    | wctp-PollForMessages
                                    | wctp-PollResponse
                                    | wctp-ReturnToSvc
                                    | wctp-SendMsgMulti
                                    | wctp-SendMsgMultiResponse
                                    | wctp-StatusInfo
                                    | wctp-SubmitClientMessage
                                    | wctp-SubmitClientResponse
                                    | wctp-SubmitRequest
                                    | wctp-VersionQuery
                                    | wctp-VersionResponse
                                    ) >

<!ATTLIST wctp-Operation
          wctpVersion               CDATA  #REQUIRED
          wctpToken                 CDATA  #IMPLIED
                                    >

<!-- =============================== -->
<!-- Declaration of wctp-ClientQuery -->
<!-- =============================== -->

<!ELEMENT wctp-ClientQuery          EMPTY
                                    >

<!ATTLIST wctp-ClientQuery
          senderID                  CDATA  #REQUIRED
          recipientID               CDATA  #REQUIRED
          trackingNumber            CDATA  #REQUIRED
                                    >

<!-- ======================================= -->
<!-- Declaration of wctp-ClientQueryResponse -->
<!-- ======================================= -->

<!ELEMENT wctp-ClientQueryResponse  ( wctp-ClientMessage+
                                   | wctp-NoMessages
                                   | wctp-Failure
                                   ) >
<!ATTLIST wctp-ClientQueryResponse
          minNextPollInterval       CDATA  #IMPLIED
                                   >

<!ELEMENT wctp-ClientMessage        ( wctp-ClientMessageReply
                                   | wctp-ClientStatusInfo
                                   ) >

<!ELEMENT wctp-ClientMessageReply   ( wctp-ClientResponseHeader
                                   , wctp-Payload
                                   ) >

<!ATTLIST wctp-ClientMessageReply
          MCRMessageReply           ( true | false ) "false"
                                   >

<!ELEMENT wctp-ClientStatusInfo     ( wctp-ClientResponseHeader
                                   , ( wctp-Failure
                                     | wctp-Notification )
                                   ) >

<!ELEMENT wctp-ClientResponseHeader ( wctp-Originator
                                   , wctp-Recipient
                                   ) >

<!ATTLIST wctp-ClientResponseHeader
          responseTimestamp         CDATA  #IMPLIED
          respondingToTimestamp     CDATA  #IMPLIED
                                   >

<!-- ================================  -->
<!-- Declaration of wctp-Confirmation  -->
<!-- ================================  -->

<!ELEMENT wctp-Confirmation         ( wctp-Failure
                                   | wctp-Success
                                   ) >

<!ELEMENT wctp-Success              ( #PCDATA )
                                   >

<!ATTLIST wctp-Success
          successCode               CDATA  #REQUIRED
          successText               CDATA  #IMPLIED
                                   >

<!-- ==================================== -->
<!-- Declaration of wctp-LookupSubscriber -->
<!-- ==================================== -->

<!ELEMENT wctp-LookupSubscriber     ( wctp-Originator
                                   , wctp-LookupMessageControl
                                   , wctp-Recipient
                                   ) >
```

```
<!ELEMENT wctp-LookupMessageControl EMPTY
                                    >

<!ATTLIST wctp-LookupMessageControl
          messageID                 CDATA  #REQUIRED
          transactionID             CDATA  #IMPLIED
          sendResponsesToID         CDATA  #IMPLIED
                                    >

<!-- ================================== -->
<!-- Declaration of wctp-LookupResponse -->
<!-- ================================== -->

<!ELEMENT wctp-LookupResponse       ( wctp-Originator
                                    , wctp-Recipient
                                    , ( wctp-LookupData
                                      | wctp-Failure
                                      )
                                    ) >

<!ATTLIST wctp-LookupResponse
          responseToMessageID       CDATA  #REQUIRED
          transactionID             CDATA  #IMPLIED
                                    >

<!ELEMENT wctp-LookupData           EMPTY
                                    >

<!ATTLIST wctp-LookupData
          maxMessageLength          CDATA  #REQUIRED
          mcrSupported              ( true | false )  #IMPLIED
          canRespond                ( true | false )  #IMPLIED
                                    >

<!-- ==================================== -->
<!-- Declaration of wctp-DeviceLocation -->
<!-- ==================================== -->

<!ELEMENT wctp-DeviceLocation     ( wctp-Originator
                                    , wctp-DeviceLocationControl
                                    , wctp-Recipient
                                    ) >

<!ELEMENT wctp-DeviceLocationControl EMPTY
                                    >

<!ATTLIST wctp-DeviceLocationControl
          messageID                 CDATA  #REQUIRED
          transactionID             CDATA  #IMPLIED
          sendResponsesToID         CDATA  #IMPLIED
          fixType                   ( approximate | estimated |
                                      precise )  “estimated”
                                    >

<!-- ================================== -->
<!-- Declaration of wctp-DeviceLocationResponse -->
```

```
<!-- ================================== -->

<!ELEMENT wctp-DeviceLocationResponse ( wctp-Originator
                                    , wctp-Recipient
                                    , ( wctp-LocationData
                                      | wctp-Failure
                                      )
                                    ) >

<!ATTLIST wctp-DeviceLocationResponse
          responseToMessageID       CDATA  #REQUIRED
          transactionID             CDATA  #IMPLIED
                                    >

<!ELEMENT wctp-LocationData           EMPTY
                                    >

<!ATTLIST wctp-LocationData
          status                    ( ok | invalid | noalt |
                                      nolonglat | stale )  #REQUIRED
          fixType                   ( approximate | estimated |
                                      precise ) #REQUIRED
          fixTimeStamp              CDATA  #IMPLIED
          longitude                 CDATA  #IMPLIED
          latitude                  CDATA  #IMPLIED
          altitude                  CDATA  #IMPLIED
                                    >


<!-- ======================================  -->
<!-- Declaration of wctp-MessageReply        -->
<!-- ======================================  -->

<!ELEMENT wctp-MessageReply         ( wctp-ResponseHeader
                                    , wctp-Payload
                                    ) >

<!ATTLIST wctp-MessageReply
          MCRMessageReply           ( true | false ) "false"
                                    >

<!ELEMENT wctp-ResponseHeader       ( wctp-Originator
                                    , wctp-MessageControl
                                    , wctp-Recipient
                                    ) >

<!ATTLIST wctp-ResponseHeader
          responseToMessageID       CDATA  #REQUIRED
          responseTimestamp         CDATA  #IMPLIED
          respondingToTimestamp     CDATA  #IMPLIED
          onBehalfOfRecipientID     CDATA  #IMPLIED
                                    >

<!-- =================================== -->
<!-- Declaration of wctp-PollForMessages -->
<!-- =================================== -->
```

```
<!ELEMENT wctp-PollForMessages      ( wctp-MessageReceived*
                                    ) >

<!ATTLIST wctp-PollForMessages
          pollerID                  CDATA  #REQUIRED
          securityCode              CDATA  #IMPLIED
          maxMessagesInBatch        CDATA  #IMPLIED
                                    >

<!ELEMENT wctp-MessageReceived      ( wctp-Failure
                                    | wctp-Success
                                    ) >

<!ATTLIST wctp-MessageReceived
          sequenceNo                CDATA  #REQUIRED
                                    >

<!-- =================================== -->
<!-- Declaration of wctp-PollResponse    -->
<!-- =================================== -->

<!ELEMENT wctp-PollResponse         ( wctp-Message+
                                    | wctp-Failure
                                    | wctp-NoMessages
                                    ) >

<!ATTLIST wctp-PollResponse
          minNextPollInterval       CDATA  #IMPLIED
                                    >

<!ELEMENT wctp-Message              ( wctp-SubmitRequest
                                    | wctp-MessageReply
                                    | wctp-StatusInfo
                                    | wctp-LookupResponse
                                    ) >

<!ATTLIST wctp-Message
          sequenceNo                CDATA  #REQUIRED
                                    >

<!ELEMENT wctp-NoMessages           EMPTY
                                    >

<!-- =================================== -->
<!-- Declaration of wctp-ReturnToSvc     -->
<!-- =================================== -->

<!--  Note: limit 1 WCTP address per wctp-ReturnToSvc  -->
<!ELEMENT wctp-ReturnToSvc          ( #PCDATA )
                                    >

<!-- ================================== -->
<!-- Declaration of wctp-SendMsgMulti   -->
<!-- ================================== -->

<!ELEMENT wctp-SendMsgMulti         ( wctp-MsgMultiHeader
                                    , wctp-Payload
                                    ) >

<!ELEMENT wctp-MsgMultiHeader       ( wctp-Originator
                                    , wctp-MsgMultiControl
                                    , wctp-Recipient+
                                    ) >

<!ATTLIST wctp-MsgMultiHeader
          submitTimestamp           CDATA  #IMPLIED
                                    >

<!ELEMENT wctp-MsgMultiControl      EMPTY
                                    >

<!ATTLIST wctp-MsgMultiControl
          messageID                 CDATA  #REQUIRED
          transactionID             CDATA  #IMPLIED
          sendResponsesToID         CDATA  #IMPLIED
          allowResponse             ( true | false ) "true"
          notifyWhenQueued          ( true | false ) "false"
          notifyWhenDelivered       ( true | false ) "false"
          notifyWhenRead            ( true | false ) "false"
          deliveryPriority          ( HIGH | NORMAL | LOW) "NORMAL"
          deliveryAfter             CDATA  #IMPLIED
          deliveryBefore            CDATA  #IMPLIED
          preformatted              ( true | false ) "false"
          allowTruncation           ( true | false ) "true"
          allRecipsRequired         ( true | false ) "false"
                                    >

<!-- ========================================= -->
<!-- Declaration of wctp-SendMsgMultiResponse  -->
<!-- ========================================= -->
<!ELEMENT wctp-SendMsgMultiResponse ( ( wctp-Failure
                                      | wctp-Success ),
                                    wctp-FailedRecipient*
                                    ) >

<!ATTLIST wctp-SendMsgMultiResponse
          maxNumRecips              CDATA  #REQUIRED
          numValidRecips            CDATA  #IMPLIED
          numInvalidRecips          CDATA  #IMPLIED
                                    >

<!ELEMENT wctp-FailedRecipient      EMPTY
                                    >

<!ATTLIST wctp-FailedRecipient
          recipientID               CDATA  #REQUIRED
        errorCode                 CDATA  #REQUIRED
          errorText                 CDATA  #IMPLIED
                                    >

<!-- ==================================  -->
<!-- Declaration of wctp-StatusInfo      -->
<!-- ==================================  -->
```

```
<!ELEMENT wctp-StatusInfo           ( wctp-ResponseHeader
                                    , ( wctp-Failure
                                      | wctp-Notification )
                                    ) >

<!ELEMENT wctp-Failure              ( #PCDATA )
                                    >

<!ATTLIST wctp-Failure
          errorCode                 CDATA  #REQUIRED
          errorText                 CDATA  #IMPLIED
                                    >

<!ELEMENT wctp-Notification         EMPTY
                                    >

<!ATTLIST wctp-Notification
          type                      ( QUEUED | DELIVERED | READ )  #REQUIRED
                                    >

<!-- ======================================= -->
<!-- Declaration of wctp-SubmitClientMessage -->
<!-- ======================================= -->

<!ELEMENT wctp-SubmitClientMessage  ( wctp-SubmitClientHeader
                                    , wctp-Payload
                                    ) >

<!ELEMENT wctp-SubmitClientHeader   ( wctp-ClientOriginator
                                    , wctp-ClientMessageControl?
                                    , wctp-Recipient
                                    ) >

<!ATTLIST wctp-SubmitClientHeader
          submitTimestamp           CDATA  #IMPLIED
                                    >

<!ELEMENT wctp-ClientOriginator     EMPTY
                                    >

<!ATTLIST wctp-ClientOriginator
          senderID                  CDATA  #REQUIRED
          miscInfo                  CDATA  #IMPLIED
                                    >

<!ELEMENT wctp-ClientMessageControl EMPTY
                                    >
<!ATTLIST wctp-ClientMessageControl
          sendResponsesToID         CDATA  #IMPLIED
          allowResponse             ( true | false ) "true"
          notifyWhenQueued          ( true | false ) "false"
          notifyWhenDelivered       ( true | false ) "false"
          notifyWhenRead            ( true | false ) "false"
          deliveryPriority          ( HIGH | NORMAL | LOW) "NORMAL"
          deliveryAfter             CDATA  #IMPLIED
          deliveryBefore            CDATA  #IMPLIED
          preformatted              ( true | false ) "false"
          allowTruncation           ( true | false ) "true"
                                    >

<!-- ======================================== -->
<!-- Declaration of wctp-SubmitClientResponse -->
<!-- ======================================== -->

<!ELEMENT wctp-SubmitClientResponse ( wctp-Failure
                                    | wctp-ClientSuccess
                                    ) >

<!ELEMENT wctp-ClientSuccess        (#PCDATA)
                                    >
<!ATTLIST wctp-ClientSuccess
          successCode               CDATA  #REQUIRED
          successText               CDATA  #IMPLIED
          trackingNumber            CDATA  #REQUIRED
                                    >

<!-- ================================== -->
<!-- Declaration of wctp-SubmitRequest  -->
<!-- ================================== -->

<!ELEMENT wctp-SubmitRequest        ( wctp-SubmitHeader
                                    , wctp-Payload
                                    ) >

<!ELEMENT wctp-SubmitHeader         ( wctp-Originator
                                    , wctp-MessageControl
                                    , wctp-Recipient
                                    ) >
<!ATTLIST wctp-SubmitHeader
          submitTimestamp           CDATA  #IMPLIED
                                    >

<!ELEMENT wctp-Originator           EMPTY
                                    >

<!ATTLIST wctp-Originator
          senderID                  CDATA  #REQUIRED
          securityCode              CDATA  #IMPLIED
          miscInfo                  CDATA  #IMPLIED
                                    >

<!ELEMENT wctp-MessageControl       EMPTY
                                    >

<!ATTLIST wctp-MessageControl
          messageID                 CDATA  #REQUIRED
          transactionID             CDATA  #IMPLIED
          sendResponsesToID         CDATA  #IMPLIED
          allowResponse             ( true | false ) "true"
          notifyWhenQueued          ( true | false ) "false"
          notifyWhenDelivered       ( true | false ) "false"
          notifyWhenRead            ( true | false ) "false"
          deliveryPriority          ( HIGH | NORMAL | LOW) "NORMAL"
          deliveryAfter             CDATA  #IMPLIED
          deliveryBefore            CDATA  #IMPLIED
          preformatted              ( true | false ) "false"
          allowTruncation           ( true | false ) "true"
                                    >

<!ELEMENT wctp-Recipient            EMPTY
                                    >
<!ATTLIST wctp-Recipient
          recipientID               CDATA  #REQUIRED
          authorizationCode         CDATA  #IMPLIED
                                    >

<!ELEMENT wctp-Payload              ( wctp-MCR
                                    | wctp-Alphanumeric
                                    | wctp-TransparentData
                                    ) >

<!ELEMENT wctp-MCR                  ( wctp-MessageText
                                    , (wctp-Choice
                                      | wctp-ChoicePair)+
                                    ) >

<!ELEMENT wctp-MessageText          ( #PCDATA )
                                    >

<!ELEMENT wctp-Choice               ( #PCDATA )
                                    >

<!ELEMENT wctp-ChoicePair           ( wctp-SendChoice
                                    , wctp-ReplyChoice? )
                                    >

<!ELEMENT wctp-SendChoice           ( #PCDATA )
                                    >

<!ELEMENT wctp-ReplyChoice          ( #PCDATA )
                                    >

<!-- ================================ -->
<!-- Declaration of wctp-VersionQuery -->
<!-- ================================ -->

<!ELEMENT wctp-VersionQuery         EMPTY
                                    >
<!ATTLIST wctp-VersionQuery
            inquirer                CDATA   #REQUIRED
            dateTime                CDATA   #IMPLIED
                                    >

<!-- ======================================== -->
<!-- Declaration of wctp-VersionResponse -->
<!-- ======================================== -->

<!ELEMENT wctp-VersionResponse      ( wctp-ContactInfo?
                                    , ( wctp-Failure
                                      | wctp-DTDSupport+ )
                                    ) >
```

```
<!ATTLIST wctp-VersionResponse
            responder               CDATA   #REQUIRED
            dateTimeOfRsp           CDATA   #IMPLIED
            inquirer                CDATA   #IMPLIED
            dateTimeOfReq           CDATA   #IMPLIED
            invalidAfter            CDATA   #IMPLIED
                                    >

<!-- =============================== -->
<!-- Declaration of wctp-ContactInfo -->
<!-- =============================== -->

<!ELEMENT wctp-ContactInfo          EMPTY
                                    >

<!ATTLIST wctp-ContactInfo
            email                   CDATA   #IMPLIED
            phone                   CDATA   #IMPLIED
            www                     CDATA   #IMPLIED
            info                    CDATA   #IMPLIED
>

<!-- ============================= -->
<!-- Declaration of wctpDTDSupport -->
<!-- ============================= -->

<!ELEMENT wctp-DTDSupport           EMPTY
                                    >

<!ATTLIST wctp-DTDSupport
          dtdName                   CDATA   #REQUIRED
          verToken                  CDATA   #IMPLIED
          supportType               ( Supported
                                    | Deprecated
                                    | NotSupported ) "Supported"
          exceptions                ( yes | no ) "no"
          supportUntil              CDATA   #IMPLIED
          replacement               CDATA   #IMPLIED
                                    >
```

```
<!-- ================================ -->
<!-- Common elements to WCTP commands -->
<!-- ================================ -->

<!ELEMENT wctp-Alphanumeric         ( #PCDATA )
                                    >

<!ELEMENT wctp-TransparentData      ( #PCDATA )
                                    >

<!ATTLIST wctp-TransparentData
          type                      ( OPAQUE
                                    | FLEXSuite
                                    | FLEXsuite ) "OPAQUE"
          encoding                  ( standard | base64 ) "base64"
                                    >
```

Appendix B The Types

This section uses the “BNF-like format” as described in Appendix C Type Summary (originally from RFC 2396, Section 1.6), to describe the valid formats for the fields. See Appendix C Type Summary for full details on the grammar and usage rules. Literal characters (characters that must appear exactly as shown) used in defining the format for the types are shown in bold type. Type names are shown in italics.

9.4 Basic Types

There are several basic types needed to describe the more complex types in WCTP. These basic types are defined here and are used in later sections.

9.4.1 Digit

Format A *Digit,* representing a single numeric character, must conform to the following format

Digit = "**0**" | "**1**" | "**2**" | "**3**" | "**4**" | "**5**" | "**6**" | "**7**" | "**8**" | "**9**"

Length *Digits* always have a length of one character.

Range The minimum equivalent integral value for a *Digit* is 0 and the maximum is 9.

Purpose To represent a single numeric character for use in other more complex types.

9.4.2 LowAlpha

Format A *LowAlpha,* representing a single lower-case alphabetic character, must conform to the following format

LowAlpha = "**a**" | "**b**" | "**c**" | "**d**" | "**e**" | "**f**" | "**g**" | "**h**" | "**i**" | "**j**" | "**k**" | "**l**" | "**m**" | "**n**" | "**o**" | "**p**" | "**q**" | "**r**" | "**s**" | "**t**" | "**u**" | "**v**" | "**w**" | "**x**" | "**y**" | "**z**"

Length *LowAlphas* always have a length of one character.

Purpose To represent a single lower-case alphabetic character for use in other more complex types.

9.4.3 UpAlpha

Format An *UpAlpha,* representing a single upper-case alphabetic character, must conform to the following format

UpAlpha = "**A**" | "**B**" | "**C**" | "**D**" | "**E**" | "**F**" | "**G**" | "**H**" | "**I**" | "**J**" | "**K**" | "**L**" | "**M**" | "**N**" | "**O**" | "**P**" | "**Q**" | "**R**" | "**S**" | "**T**" | "**U**" | "**V**" | "**W**" | "**X**" | "**Y**" | "**Z**"

Length *UpAlphas* always have a length of one character.

Purpose To represent a single upper-case alphabetic character for use in other more complex types.

9.4.4 Alpha

Format An *Alpha,* representing a single lower- or upper-case alphabetic character, must conform to the following format

Alpha = *LowAlpha* |*UpAlpha*

Length *Alphas* always have a length of one character.

Purpose To represent a single upper-case alphabetic character for use in other more complex types.

9.4.5 AlphaNum

Format An *AlphaNum,* representing a single alphanumeric character, must conform to the following format

AlphaNum = *Alpha* |*Digit*

Length *AlphaNums* always have a length of one character.

Purpose To represent a single alphanumeric character in other more complex types.

9.4.6 NegativeNumber

Format A *NegativeNumber*, a series of characters that together represent a negative (less than zero) integer value, must conform to the following format

NegativeNumber = "**-**" ("**1**" | "**2**" | "**3**" | "**4**" | "**5**" | "**6**" | "**7**" | "**8**" | "**9**") **Digit*

Length *NegativeNumbers* have a minimum length of two characters; the maximum length is unspecified, but should be stated for each attribute that uses this type.

Purpose To define a negative integer type.

9.4.7 PositiveNumber

Format A *PositiveNumber*, a series of characters that together represent a positive (greater than zero) integer value, must conform to the following format

PositiveNumber = ("**1**" | "**2**" | "**3**" | "**4**" | "**5**" | "**6**" | "**7**" | "**8**" | "**9**") **Digit*

Length *PositiveNumbers* have a minimum length of one character; the maximum length is unspecified, but should be stated for each attribute that uses this type.

Purpose To define a positive integer type.

9.4.8 NonPositiveNumber

Format A *NonPositiveNumber*, a series of characters that together represent a non-positive (less than or equal to zero) integer value, must conform to the following format

NonPositiveNumber = *NegativeNumber* | "**0**"

Length *NonPositiveNumbers* have a minimum length of one character; the maximum length is unspecified, but should be stated for each attribute that uses this type.

Purpose To define a non-positive integer type.

9.4.9 NonNegativeNumber

Format A *NonNegativeNumber*, a series of characters that together represent a non-negative (greater than or equal to zero) integer value, must conform to the following format

NonNegativeNumber = "**0**" | *PositiveNumber*

Length *NonNegativeNumbers* have a minimum length of one character; the maximum length is unspecified, but should be stated for each attribute that uses this type.

Purpose To define a non-negative integer type.

9.4.10 Number

Format A *Number*, a series of characters that together represent an integer value, must conform to the following format

Number = *NegativeNumber* | "**0**" | *PositiveNumber*

Length *Numbers* have a minimum length of one character; the maximum length is unspecified, but should be stated for each attribute that uses this type.

Purpose To define an integer type.

9.4.11 ByteNum

Format A *ByteNum,* representing an integer value that can be stored in one byte (eight bits), must conform to the following format

ByteNum = [["**1**" | "**2**"] *Digit*] *Digit*

Length *ByteNums* have a minimum length of one character and a maximum length of three characters.

Range The minimum equivalent integral value for a *ByteNum* is 0 and the maximum is 255.

Purpose To represent an integer value between 0 and 255 for use in other more complex types.

9.4.12 Hex

Format A *Hex,* representing a single hexadecimal character, must conform to the following format

Hex = *Digit* | "**A**" | "**B**" | "**C**" | "**D**" | "**E**" | "**F**" | "**a**" | "**b**" | "**c**" | "**d**" | "**e**" | "**f**"

Length *Hexes* always have a length of one character.

Range The minimum equivalent integral value for a *Hex* is 0 and the maximum is 15.

Purpose To represent a single hexadecimal character for use in other more complex types.

9.4.13 Escaped

Format An *Escaped,* an integer value that can be stored in one byte (eight bits) expressed in hexadecimal format, must conform to the following format

Escaped = ("**&**" "**a**" "**m**" "**p**" ";") | ("**&**" "**l**" "**t**" ";") | ("**&**" "**g**" "**t**" ";") | ("**&**" "**a**" "**p**" "**o**" "**s**" ";") | ("**&**" "**q**" "**u**" "**o**" "**t**" ";") |("**&**" "#" "**x**" *Hex Hex Hex Hex* ";")

Length *Escaped* types have a minimum length of four characters and a maximum length of eight characters.

Range The minimum equivalent integral value for an *Escaped* is 0 and the maximum is 255.

Purpose To represent an integer value between 0 and 255 in hexadecimal-character format for use in other more complex types. It corresponds to the standard XML Encoding format, and as such the only valid values for the hexadecimal values represented by an Escaped character are the Horizontal Tab (x0009), Line Feed (x0010), Carriage Return (x0013), and the printable characters between x0020 and x007E (decimal 32 and 126), inclusive.

9.4.14 Mark

Format A *Mark,* representing a single printable non-alphanumeric character, must conform to the following format

Mark = "-" | "_" | "." | "!" | "~" | "*" | "(" | ")" | "\" | "/" | "#" | "$" | "%" | "?" | "=" | "+"

Length *Marks* always have a length of one character.

Purpose To represent a single printable non-alphanumeric character for use in other more complex types.

9.4.15 Unreserved

Format An *Unreserved,* representing a single printable character that is available for common use, must conform to the following format

Unreserved = Mark | *AlphaNum*

Length *Unreserveds* always have a length of one character.

Purpose To represent a single printable character that is available for common use in other more complex types.

9.4.16 PathChar

Format A *PathChar,* representing a single character that is valid in directory paths, must conform to the following format

PathChar =   Unreserved | *Escaped*

Length *PathChars* always have a length of one character.

Purpose To represent a single character that is valid in directory paths.

9.4.17 WCTPChar

Format A *WCTPChar,* representing a single character that is valid in most WCTP fields, must conform to the following format

WCTPChar = Unreserved | *Escaped* | "**;**" | "@" | "**|**" | "**,**" | "**[**" | "**]**" | ":" | "**{**" | "**}**" | "**^**"

Length *WCTPChars* always have a length of one character.

Purpose To represent a single character that is valid in directory paths.

9.4.18 String

Format A *String,* representing a string of characters that is valid in many WCTP fields, must conform to the following format

String = 1**WCTPChar*

Length *Strings* have a minimum length of one character; the maximum length is unspecified, but should be stated for each attribute that uses this type.

Purpose To represent a character string that is valid in most WCTP attributes

.

9.4.19 CapString

Format A *CapString,* representing a string of digits or upper case letters, must conform to the following format

CapString = 1*(*UpAlpha* | *Digit*)

Length _CapStrings_ have a minimum length of one character; the maximum length is unspecified, but should be stated for each attribute that uses this type.

Purpose To represent an upper-case character string.

9.4.20 Version

Format A _Version,_ representing a string of characters that make up a valid WCTP version identifier, must conform to the following format

Version = _CapString_ "**-**" _CapString_ "**-**" "**V**" 1*_Digit_ "**R**" 1*_Digit_

Length _Versions_ have a minimum length of eight characters, and a maximum length of twenty-four characters.

Purpose To represent a character string used for WCTP version identifiers.

9.4.21 PassCode

Format A _PassCode,_ representing a string of characters, must conform to the following format

PassCode = 1*_Unreserved_

Length _PassCodes_ have a minimum length of one character; the maximum length is unspecified, but should be stated for each attribute that uses this type.

Purpose To represent a password character string.

9.4.22 PhoneNumber

Format A _PhoneNumber_, representing a valid telephone number, must conform to the following format

PhoneNumber = _String_

Length _PhoneNumbers_ have a minimum length of ten characters, and a maximum length of thirty-two characters.

Purpose To represent a valid telephone number.

9.5 Enumerated Types

There are several enumerated types needed for fields in the WCTP. These enumerated types are defined here.

9.5.1 TrueFalse

Format A *TrueFalse,* representing a case insensitive enumeration of character strings, must conform to the following format

TrueFalse = ("**t**" "**r**" "**u**" "**e**") | ("**f**" "**a**" "**l**" "**s**" "**e**")

Length *TrueFalses* have a length of either four or five characters.

Purpose To represent the case-insensitive values 'true' and 'false' as the only valid options for a field.

9.5.2 YesNo

Format A *YesNo,* representing a case insensitive enumeration of character strings, must conform to the following format

YesNo = ("**y**" "**e**" "**s**") | ("**n**" "**o**")

Length *YesNos* have a length of either two or three characters.

Purpose To represent the case-insensitive values 'yes' and 'no' as the only valid options for a field.

9.5.3 DeliveryPriority

Format A *DeliveryPriority,* representing a case insensitive enumeration of character strings, must conform to the following format

DeliveryPriority = ("**H**" "**I**" "**G**" "**H**") | ("**N**" "**O**" "**R**" "**M**" "**A**" "**L**") | ("**L**" "**O**" "**W**")

Length *DeliveryPrioritys* have a minimum length of three characters and a maximum length of six characters.

Purpose To represent the case-insensitive values 'high', 'normal', and 'low' as the only valid options for a field.

9.5.4 Notification

Format A *Notification,* representing a case insensitive enumeration of character strings, must conform to the following format

Notification = ("**Q**" "**U**" "**E**" "**U**" "**E**" "**D**") | ("**D**" "**E**" "**L**" "**I**" "**V**" "**E**" "**R**" "**E**" "**D**") | ("**R**" "**E**" "**A**" "**D**")

Length *Notifications* have a minimum length of four characters and a maximum length of nine characters.

Purpose To represent the case-insensitive values 'queued', 'delivered', and 'read' as the only valid options for a field.

9.5.5 SupportType

Format A *SupportType,* representing a case insensitive enumeration of character strings, must conform to the following format

SupportType = ("**S**" "**u**" "**p**" "**p**" "**o**" "**r**" "**t**" "**e**" "**d**") | ("**D**" "**e**" "**p**" "**r**" "**e**" "**c**" "**a**" "**t**" "**e**" "**d**") | ("**N**" "**o**" "**t**" "**S**" "**u**" "**p**" "**p**" "**o**" "**r**" "**t**" "**e**" "**d**")

Length *SupportTypes* have a minimum length of nine characters and a maximum length of twelve characters.

Purpose To represent the case-insensitive values 'Supported', 'Deprecated', and 'NotSupported' as the only valid options for a field.

9.5.6 DataType

Format A *DataType,* representing a case insensitive enumeration of character strings, must conform to the following format

DataType = (“**O**” “**P**” “**A**” “**Q**” “**U**” “**E**”) | (“**F**” “**L**” “**E**” “**X**” “**S**” “**u**” “**i**” “**t**” “**e**”)

Length *DataTypes* have a minimum length of six characters and a maximum length of nine characters.

Purpose To represent the case-insensitive values ‘OPAQUE’ and ‘FLEXSuite’ as the only valid options for a field.

9.5.7 EncodingType

Format An *EncodingType,* representing a case insensitive enumeration of character strings, must conform to the following format

EncodingType = ("**s**" "**t**" "**a**" "**n**" "**d**" "**a**" "**r**" "**d**") | ("**b**" "**a**" "**s**" "**e**" "**6**" "**4**")

Length *EncodingTypes* have a minimum length of six characters and a maximum length of eight characters.

Purpose To represent the case-insensitive values 'standard' and 'base64' as the only valid options for a field.

9.5.8 FailReasonType

Format A *FailReasonType,* representing a case insensitive enumeration of character strings, must conform to the following format

FailReasonType = ("**u**" "**n**" "**k**" "**n**" "**o**" "**w**" "**n**") | ("**t**" "**e**" "**m**" "**p**" "**F**" "**a**" "**i**" "**l**" "**u**" "**r**" "**e**") | ("**i**" "**n**" "**v**" "**a**" "**l**" "**i**" "**d**" "**A**" "**u**" "**t**" "**h**" "**C**" "**o**" "**d**" "**e**")

Length *FailReasonTypes* have a minimum length of seven characters and a maximum length of fifteen characters.

Purpose To represent the case-insensitive values 'unknown', 'tempFailure', and 'invalidAuthCode' as the only valid options for a field.

9.6 Address Types

Section 2.4 of the WCTP 1.1 Protocol Specification defines an address by specifying the format for entity-addresses and transport addresses. It also provides examples of valid addresses for wireless devices and wireline hosts. The formats specified there for an address, entity-address, and transport-address are used as a base for the address types defined here.

All address types shown here must contain only characters that comply with RFC 2396, section 2, *URI Characters*.

9.6.1 Alias

Format An *Alias* must conform to the following format

*Alias = *(“!”)*AlphaNum 1*Alpha *AlphaNum*

Length The minimum length is one character, and the maximum length is thirty-two characters.

Purpose To define a shorter substitute for an address.

9.6.2 PollerID

Format A *PollerID* must conform to the following format

*PollerID = 1*AlphaNum "**.**" 1*AlphaNum*

Length The minimum length is three characters, and the maximum length is one hundred twenty-eight characters.

Purpose To define an identifier for an enterprise organization to use wctp-PollForMessages.

9.6.3 Protocol

Format A *Protocol* must conform to the following format

Protocol = Alpha *(*Alpha* | *Digit* | "**+**" | "**-**" | "**.**")

Length The minimum length is one character, and the maximum length is eight characters.

Purpose To specify the protocol to be used in transporting a WCTP operation. Some common examples include "HTTP" and "FTP".

9.6.4 IPAddress

Format An *IPAddress* must conform to the following format

IPAddress = ByteNum "**.**" *ByteNum* "**.**" *ByteNum* "**.**" *ByteNum*

Length The minimum length is seven characters and the maximum length is fifteen characters.

Purpose To specify the IP Address version of a domain name.

9.6.5 TopLabel

Format A *TopLabel* must conform to the following format

TopLabel = Alpha | *Alpha* *(*AlphaNum* | "**-**") *AlphaNum*

Length The minimum length is one character and the maximum length is eight characters.

Purpose To specify the top level portion or segment of a domain name.

9.6.6 DomainLabel

Format A *DomainLabel* must conform to the following format

DomainLabel = AlphaNum | *AlphaNum* *(*AlphaNum* | "**-**") *AlphaNum*

Length The minimum length is one character and the maximum length is forty-eight characters.

Purpose To specify a portion or segment of a domain name other than the top level.

9.6.7 HostName

Format A *HostName* must conform to the following format

HostName = 1*(*DomainLabel* "**.**") *TopLabel*

Length The minimum length is three characters and the maximum length is one hundred twenty-eight characters. NOTE: HostName may be used as a component of TransportAddress and Address which are also constrained to one hundred twenty-eight characters. When used as a component of another field, the length of HostName must be limited such that the total field length is no greater than one hundred twenty-eight characters.

Purpose To specify a non-IP address domain name for a computer.

9.6.8 Domain

Format A *Domain* must conform to the following format

Domain = HostName | *IPAddress*

Length The minimum length is three characters and the maximum length is one hundred twenty-eight characters. NOTE: Domain may be used as a component of TransportAddress and Address which are also constrained to one hundred twenty-eight characters. When used as a component of another field, the length of Domain must be limited such that the total field length is no greater than one hundred twenty-eight characters.

Purpose To specify an actual individual or application by name.

9.6.9 Port

Format A *Port,* representing an integer, must conform to the following format

Port = [[[["**1**" | "**2**" | "**3**" | "**4**" | "**5**" | "**6**"] *Digit*] *Digit*] *Digit*] *Digit*

Length The minimum length is one character and the maximum length is five characters.

Range The minimum numeric value is 1 and the maximum value is 65535.

Purpose To specify a socket port involved in an address for a WCTP operation.

9.6.10 PathSegment

Format A *PathSegment* must conform to the following format

*PathSegment = 1*PathChar* "**/**"

Length The minimum length is one character, and the maximum length is sixteen characters.

Purpose To specify a segment of a directory path.

9.6.11 Path

Format A *Path* must conform to the following format

Path = "**/**" *(*PathSegment) *PathChar*

Length The minimum length is one character, and the maximum length is one hundred twenty-eight characters. NOTE Path may be used as a component of TransportAddress, EntityAddress, and Address, which are also constrained to one hundred twenty-eight characters. When used as a component of another field, the length of Path must be limited such that the total field length is no greater than one-hundred twenty-eight characters.

Purpose To specify an absolute path to an application to accept the operation.

9.6.12 TransportAddress

Format A *TransportAddress* must conform to the following format

TransportAddress = [*Protocol* ":" *Port*] [*Path*]

Length The minimum length is three characters, and the maximum length is one hundred twenty-eight characters.

Purpose To specify the full “transport protocol” portion of an address to be used in a WCTP operation. This generally represents a server on a network, including how to locate it and how to communicate with it.

9.6.13 Scheme

Format A *Scheme* must conform to the following format

Scheme = Alpha *(*Alpha* | *Digit* | "**+**" | "**-**" | "**.**")

Length The minimum length is one character, and the maximum length is eight characters.

Purpose To specify the communications scheme to be used in an address for a WCTP operation. Some common examples include "phoneto" and "faxto".

9.6.14 Entity

Format An *Entity* must conform to the following format

Entity = *1*WCTPChar*

Length The minimum length is one character, and the maximum length is thirty-two characters.

Purpose To specify an actual individual or application by name.

9.6.15 EntityAddress

Format An *EntityAddress* must conform to the following format

EntityAddress = [*Scheme* ":" *Port*][*Path*]

Length The minimum length is one character, and the maximum length is one hundred twenty-eight characters.

Purpose To specify the full “entity” portion of an address to be used in a WCTP operation. This generally represents an individual or application, including how to locate that entity.

9.6.16 Address

Format An *Address* must conform to the following format

Address = ([*EntityAddress* "@"] *TransportAddress*) | *Alias* | *PollerID*

Length The minimum length is one character, and the maximum length is one hundred twenty-eight characters.

Purpose To specify an address needed in a WCTP operation, such as a recipient or sender.

9.6.17 EmailAddress

Format An *EmailAddress* in the WCTP must conform to the following format

EmailAddress = *Entity* "@" *Domain*

Length The minimum length is five characters, and the maximum length is one hundred twenty-eight characters.

Purpose To specify a full SMTP-compliant email address to be used in WCTP operations.

9.6.18 WWWAddress

Format A *WWWAddress* must conform to the following format

WWWAddress = Domain [Path]

Length The minimum length is three characters and the maximum length is one hundred twenty-eight characters.

Purpose To specify a valid World-Wide-Web address to be used in a wctp-ContactInfo element. This type of address represents a web server that can be accessed by using a standard browser.

9.7 Date and Time Types

Section 3.2 of the WCTP 1.1 Protocol Specification defines the valid date, time, and datetime formats. It also provides examples of valid timestamps for WCTP operations. The formats specified there for a date, time and datetime stamp are used as a base for the date and time types defined here.

9.7.1 Day

Format A *Day*, a two-digit value representing a calendar day, must conform to the following format

Day = ("**0**" ("**1**" | "**2**" | "**3**" | "**4**" | "**5**" | "**6**" | "**7**" | "**8**" | "**9**")) | ("**1**" *Digit*) | ("**2**" *Digit*) | ("**3**" ("**0**" | "**1**"))

If the *Day* is associated with a *Month*, then the set of valid values may be further restricted to match the number of days in that month according to the Gregorian calendar, including taking into account leap years. For example, in a non-leap year such as 1999, the values "**2**" "**9**", "**3**" "**0**", and "**3**" "**1**" are NOT valid for the *Month* of "**0**" "**2**" (February).

Length A *Day* must always have a length of two characters.

Purpose To define a type that represents a valid day in a calendar month.

9.7.2 Month

Format A *Month*, a two-digit value representing a calendar month, must conform to the following format

Month = ("**0**" ("**1**" | "**2**" | "**3**" | "**4**" | "**5**" | "**6**" | "**7**" | "**8**" | "**9**")) | ("**1**" ("**0**" | "**1**" | "**2**"))

The value for January is "**0**" "**1**", for February "**0**" "**2**", and so on through December as "**1**" "**2**".

Length A *Month* must always have a length of two characters.

Purpose To define a type that represents a valid month in a calendar year.

9.7.3 Year

Format A *Year*, a four-digit value representing a calendar year, must conform to the following format

Year = *Digit Digit Digit Digit*

A *Year* has a set of valid values ranging from "**0**" "**0**" "**0**" "**0**" to "**9**" "**9**" "**9**" "**9**".

Length A *Year* must always have a length of four characters.

Purpose To define a type that represents a valid year in the Gregorian calendar.

9.7.4 Date

Format A *Date*, a series of characters that represent a year, month, and day, must conform to the following format

Date = *Year* "**-**" *Month* "**-**" *Day*

Length A *Date* must always have a length of ten characters.

Purpose To define a type that represents a valid date in the Gregorian calendar.

9.7.5 Seconds

Format *Seconds*, a series of characters that represent the number of seconds past a particular minute, must conform to the following format

Seconds = ("**0**" | "**1**" | "**2**" | "**3**" | "**4**" | "**5**") *Digit*

Seconds has a set of valid values ranging from "**0**" "**0**" to "**5**" "**9**".

Length *Seconds* must always have a length of two characters.

Purpose To define a type that represents a valid number of seconds past a minute.

9.7.6 Minutes

Format *Minutes*, a series of characters that represent the number of minutes past a particular hour, must conform to the following format

Minutes = ("**0**" | "**1**" | "**2**" | "**3**" | "**4**" | "**5**") *Digit*

Minutes has a set of valid values ranging from "**0**" "**0**" to "**5**" "**9**".

Length *Minutes* must always have a length of two characters.

Purpose To define a type that represents a valid number of minutes past an hour.

9.7.7 Hours

Format *Hours*, a series of characters that represent the number of hours past midnight on a particular day, must conform to the following format

Hours = ("**0**" *Digit*) | ("**1**" *Digit*) | ("**2**" ("**0**" | "**1**" | "**2**" | "**3**"))

Hours has a set of valid values ranging from "**0**" "**0**" to "**2**" "**3**".

Length *Hours* must always have a length of two characters.

Purpose To define a type that represents a valid number of hours past midnight in a particular day.

9.7.8 Time

Format A *Time*, a series of characters that represent the number of hours, minutes, and seconds (along with an optional subseconds value) past midnight on a particular day, must conform to the following format

Time = ("**2**" "**4**" ":" "**0**" "**0**" ":" "**0**" "**0**") | (*Hours* ":" *Minutes* "”" *Seconds* ["**,**" *Digit* [*Digit* [*Digit*]]])

Time includes the value of '24:00:00', which is equivalent to '00:00:00', and both of which represent midnight, but potentially on the boundary between different dates. For example, '24:00:00' on June 3rd is NOT equivalent to '00:00:00' on June 3rd, but rather is equivalent to '00:00:00' on June 4th.

Length A *Time* has a minimum length of eight characters and a maximum length of twelve characters.

Purpose To define a type that represents a valid time during a day.

9.7.9 DateTime

Format A *DateTime*, a series of characters that represent a year, month, and day, together with a time expressed as hours, minutes, and seconds, must conform to the following format

DateTime = *Date* "**T**" *Time*

Length A *DateTime* has a minimum length of nineteen characters and a maximum length of twenty-three characters.

Purpose To define a type that represents a valid date and time in the Gregorian calendar.

9.7.10 TimeInterval

Format A *TimeInterval*, a series of characters representing an integer as a period of elapsed time in seconds, must conform to the following format

TimeInterval = NonNegativeNumber

Length A *TimeInterval* has a minimum length of one character; the maximum length is unspecified, but should be stated for each attribute that uses this type.

Range The minimum numeric value is 0; the maximum numeric value should be stated for each attribute that uses this type.

Purpose To define a type that represents a valid time interval in seconds.

9.8 Location Data Types

9.8.1 Location-Degrees

Format *Location-Degrees* is a series of digits that represent global location in degrees longitude or latitude.

Location-Degrees =

("**0**””**0**" *Digit*) | ("**0**" *Digit Digit*) | (*Digit Digit Digit*)

Location-Degrees has a set of valid values ranging from "**0**" "**0**"”**0**” to "**3**" "**5**"”**9”**.

Length *Location-Degrees* must always have a length of three characters.

Purpose To define a type that represents a valid number of degrees longitude or latitude.

9.8.2 Location-Minutes

Format *Location-Minutes* is a series of digits that represent the minutes portion of global location in either longitude or latitude.

Location-Minutes =

("**0**" *Digit*) | (*Digit Digit*)

Location-Minutes has a set of valid values ranging from "**0**" "**0**" to "**5**" "**9**".

Length Location-Minutes must always have a length of two characters.

Purpose To define a type that represents a valid number of minutes in the global position in longitude or latitude.

9.8.3 Location-Seconds

Format *Location-Seconds* is a series of digits that represent the seconds portion of global location in either longitude or latitude. These digits encode the fractional portion of a minute.

Location-Seconds =

("**0**”"**0**""**0**” *Digit*) | ("**0**”"**0**" *Digit Digit*) |
("**0**” *Digit Digit Digit*) | (*Digit Digit Digit Digit*)

Location-Seconds has a set of valid values ranging from "**0**" "**0**""**0**" "**0**" to "**9**" "**9**""**9**" "**9**"..

Length Location-Seconds must always have a length of four characters.

Purpose To define a type that represents a valid fraction of a minute in order to represent the number of seconds in the global position in longitude or latitude. Accuracy required is from zero to four digits.

9.8.4 Location

Format *Location* is a number representing either the longitude or latitude in a format similar used by the NMEA for GPS and must conform to the following format.

An example of a valid location is “12301.3300W” which represents 123 degrees 1.33 minutes West longitude.

An example of a valid location where the degrees portion is less than 100 is “05145.3211W” which represents 51 degrees 45.3211 minutes North latitude

Location = Location-Degrees Location-Minutes ”**.**” Location-Seconds (“**N**”|”**S**”|”**E**”|”**W**”)

Length *Location* must always have a length of eleven characters.

Purpose To define a type that represents a valid global location in longitude or latitude.

Appendix C Type Summary

This section uses a BNF-like grammar as defined in RFC2396. Many of the details presented here are similar to those defined in RFC2396, although some have been changed to meet the needs of the WCTP.

Briefly, rules are separated from definitions by an equal "=", indentation is used to continue a rule definition over more than one line, literals are quoted with "", parentheses "(" and ")" are used to group elements, optional elements are enclosed in "[" and "]" brackets, "|" is used to designate alternatives, and elements may be preceded with n* to designate n or more repetitions of the following element, where n is optional and defaults to 0.

Unlike many specifications that use a BNF-like grammar to define the bytes (octets) allowed by a protocol, this grammar is defined in terms of characters. Each literal in the grammar corresponds to the character it represents, rather than to the octet encoding of that character in any particular coded character set. How WCTP operations are represented in terms of bits and bytes on the wire is dependent upon the character encoding of the protocol used to transport it, or the character set of the document that contains it.

9.9 Basic Types

The following basic field types are used in multiple field format definitions.

```
Digit              =     "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                         "8" | "9"
LowAlpha           =     "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" |
                         "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" |
                         "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" |
                         "y" | "z"
UpAlpha            =     "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" |
                         "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" |
                         "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" |
                         "Y" | "Z"
Alpha              =     LowAlpha | UpAlpha
AlphaNum           =     Alpha | Digit
NegativeNumber     =     "-" ( "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                         "8" | "9" ) *Digit
PositiveNumber     =     ( "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8"
                         | "9" ) *Digit
NonPositiveNumber  =     NegativeNumber | "0"
NonNegativeNumber  =     "0" | PositiveNumber
Number             =     NegativeNumber | "0" | PositiveNumber
ByteNum            =     [ [ "1" | "2" ] Digit ] Digit
Hex                =     Digit | "A" | "B" | "C" | "D" | "E" | "F" | "a"
                         | "b" | "c" | "d" | "e" | "f"
Escaped            =     ( "&" "a" "m" "p" ";" ) | ( "&" "l" "t" ";" ) |
                         ( "&" "g" "t" ";" ) | ( "&" "a" "p" "o" "s" ";"
                         ) | ( "&" "q" "u" "o" "t" ";" ) |( "&" "#" "x"
                         Hex Hex Hex Hex ";" )
Mark               =     "-" | "_" | "." | "!" | "~" | "*" | "(" | ")" |
                         "\" | "/" | "#" | "$" | "%" | "?" | "=" | "+"
Unreserved         =     Mark | AlphaNum
PathChar           =     Unreserved | Escaped
WCTPChar           =     Unreserved | Escaped | ";" | "@" | "|" | "," |
                         "[" | "]" | ":" | "{" | "}" | "^"
String             =     1*WCTPChar
CapString          =     1*( UpAlpha | Digit )
Version            =     CapString "-" CapString "-" "V" 1*Digit "R"
                         1*Digit
PassCode           =     1*Unreserved
PhoneNumber        =     String
```

9.10 Enumerated Types

The following complex types represent the enumerated field types used in the WCTP field format definitions.

```
TrueFalse         =     ( "t" "r" "u" "e" ) | ( "f" "a" "l" "s" "e" )
YesNo             =     ( "y" "e" "s" ) | ( "n" "o")
DeliveryPriority  =     ( "H" "I" "G" "H" ) | ( "N" "O" "R" "M" "A" "L"
                        ) | ( "L" "O" "W" )
Notification      =     ( "Q" "U" "E" "U" "E" "D" ) | ( "D" "E" "L" "I"
                        "V" "E" "R" "E" "D" ) | ( "R" "E" "A" "D" )
SupportType       =     ( "S" "u" "p" "p" "o" "r" "t" "e" "d" ) | ( "D"
                        "e" "p" "r" "e" "c" "a" "t" "e" "d" ) | ( "N"
                        "o" "t" "S" "u" "p" "p" "o" "r" "t" "e" "d" )
DataType          =     ( "O" "P" "A" "Q" "U" "E" ) | ( "F" "L" "E" "X"
                        "S" "u" "i" "t" "e" )
EncodingType      =     ( "s" "t" "a" "n" "d" "a" "r" "d" ) | ( "b" "a"
                        "s" "e" "6" "4" )
FailReasonType    =     ( "u" "n" "k" "n" "o" "w" "n" ) | ( "t" "e" "m"
                        "p" "F" "a" "i" "l" "u" "r" "e" ) | ( "i" "n"
                        "v" "a" "l" "i" "d" "A" "u" "t" "h" "C" "o" "d"
                        "e" )
```

9.11 Address Types

The following complex types represent the more detailed and complicated field types used in the WCTP address field format definitions.

```
Alias             =     *(“!”)*AlphaNum 1*Alpha *AlphaNum

PollerID          =     1*AlphaNum "." 1*AlphaNum

Protocol          =     Alpha *( Alpha | Digit | "+" | "-" | "." )

IPAddress         =     ByteNum "." ByteNum "." ByteNum "." ByteNum

TopLabel          =     Alpha | Alpha *( AlphaNum | "-" ) AlphaNum

DomainLabel       =     AlphaNum | AlphaNum *( AlphaNum | "-" )
                        AlphaNum

HostName          =     1*( DomainLabel "." ) TopLabel

Domain            =     HostName | IPAddress

Port              =     [ [ [ [ "1" | "2" | "3" | "4" | "5" | "6" ]
                        Digit ] Digit ] Digit ] Digit

PathSegment       =     1*PathChar "/"

Path              =     "/" *( PathSegment ) *PathChar

TransportAddress  = [ Protocol ":" [ "/" "/" ] ] Domain [ ":" Port ] [
                        Path ]

Scheme            =     Alpha *( Alpha | Digit | "+" | "-" | "." )

Entity            =     1*WCTPChar

EntityAddress     = [ Scheme ":" ] Entity [":" Port ][ Path ]

Address           = [ EntityAddress "@" ] TransportAddress

EmailAddress      = Entity "@" Domain

WWWAddress        = Domain [ Path ]
```

9.12 Date and Time Types

The following complex types represent the more detailed and complicated field types used in the WCTP date and time field format definitions.

```
Day                 =     ( "0" ( "1" | "2" | "3" | "4" | "5" | "6" | "7"
                          | "8" | "9" ) ) | ( "1" Digit ) | ( "2" Digit )
                          | ( "3" ( "0" | "1" ) )
Month               =     ( "0" ( "1" | "2" | "3" | "4" | "5" | "6" | "7"
                          | "8" | "9" ) ) | ( "1" ( "0" | "1" | "2" ) )
Year                =     Digit Digit Digit Digit
Date                =     Year "-" Month "-" Day
Seconds             =     ( "0" | "1" | "2" | "3" | "4" | "5" ) Digit
Minutes             =     ( "0" | "1" | "2" | "3" | "4" | "5" ) Digit
Hours               =     ( "0" Digit ) | ( "1" Digit ) | ( "2" ( "0" |
                          "1" | "2" | "3" ) )
Time                =     ( "2" "4" ":" "0" "0" ":" "0" "0" ) | ( Hours
                          ":" Minutes ":" Seconds [ "," Digit [ Digit [
                          Digit ] ] ] )
DateTime            =     Date "T" Time
TimeInterval        =     1*Digit
```

9.13 Location Types

```
Location-Degrees    =     ("0" "0" Digit) |
                          (("0" | "1" | "2")Digit Digit) |
                          ("3" ("0" | "1" | "2" | "3" | "4" | "5") Digit)
Location-Minutes    =     ( "0" | "1" | "2" | "3" | "4" | "5" ) Digit
Location-Seconds    =     Digit Digit Digit Digit
Location            =     Location-Degrees Location-Minutes "." Location-
                          Seconds ("N" | "S" | "E" | "W")
```

Appendix D Attribute Definitions

Each attribute in the WCTP protocol is described in the below sections. Whenever possible, a field is described using one of the data types defined earlier.

The column titled "Attribute Name" gives the XML attribute name from the WCTP DTD.

The column titled "Disposition" refers to whether the attribute is Optional, Mandatory (Required), or Conditional.

The column titled "Type" gives the date type on which this attribute is based.

The column titled "Range" refers to the valid range of numeric values for the attribute. This column is only applicable when the attribute represents a numeric value. Otherwise, it contains "N/A" for "Not Applicable."

The column titled "Length" defines the minimum and maximum number of characters that can be placed in the attribute whenever it is present.

The column titled "Default" defines the default value for the attribute (if any). This column is only applicable when the attribute represents an attribute with an optional value in the DTD. Otherwise, it contains "N/A" for "Not Applicable."

9.14 wctp-Operation

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
`wctpVersion`	Mandatory	Version	N/A	8 to 24	N/A	3.6
`wctpToken`	Optional	String	N/A	1 to 16	N/A	3.7

9.15 wctp-SubmitHeader

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
`submitTimestamp`	Mandatory	DateTime	N/A	19 to 23	N/A	7.1.3, 5.1.3

9.16 wctp-Originator

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
`senderID`	Mandatory	Address	N/A	1 to 128	N/A	7.1.1
`securityCode`	Optional	PassCode	N/A	1 to 16	N/A	7.1.1
`MiscInfo`	Optional	String	N/A	1 to 16	N/A	7.1.1

9.17 wctp-MessageControl

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
messageID	Mandatory	String	N/A	1 to 32	N/A	8.2.1.1
transactionID	Optional	String	N/A	1 to 32	N/A	8.2.1.1
sendResponsesToID	Optional	Address	N/A	1 to 128	N/A	7.1.4.1
allowResponse	Optional	TrueFalse	N/A	4 to 5	true	7.1.4.2
notifyWhenQueued	Optional	TrueFalse	N/A	4 to 5	false	7.1.4.3
notifyWhenDelivered	Optional	TrueFalse	N/A	4 to 5	false	7.1.4.4
notifyWhenRead	Optional	TrueFalse	N/A	4 to 5	false	7.1.4.5
deliveryPriority	Optional	DeliveryPriority	N/A	3 to 6	NORMAL	7.1.4.6
deliveryBefore	Optional	DateTime	N/A	19 to 23	N/A	7.1.4.7
deliveryAfter	Optional	DateTime	N/A	19 to 23	N/A	7.1.4.8
preformatted	Optional	DateTime	N/A	4 to 5	false	7.1.4.9
allowTruncation	Optional	DateTime	N/A	4 to 5	true	7.1.4.10

9.18 wctp-Recipient

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
recipientID	Mandatory	Address	N/A	1 to 128	N/A	7.1.2
authorizationCode	Optional	PassCode	N/A	1 to 16	N/A	7.1.2

9.19 wctp-Alphanumeric

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
(PCDATA)	Optional	String	N/A	1 to 65535	N/A	7.2.1

9.20 wctp-TransparentData

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
(PCDATA)	Optional	String	N/A	1 to 65535	N/A	7.2.3
type	Optional	DataType	N/A	6 to 9	OPAQUE	7.2.3
encoding	Optional	EncodingType	N/A	6 to 8	base64	7.2.3

9.21 wctp-MessageText

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
(PCDATA)	Optional	String	N/A	1 to 65535	N/A	7.2.2

9.22 wctp-Choice

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
(PCDATA)	Optional	String	N/A	1 to 32	N/A	7.2.2

9.23 wctp-SendChoice

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
(PCDATA)	Optional	String	N/A	1 to 32	N/A	7.2.2

9.24 wctp-ReplyChoice

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
(PCDATA)	Optional	String	N/A	1 to 64	N/A	7.2.2

9.25 wctp-MessageReply

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
MCRReplyChoice	Optional	TrueFalse	N/A	4 to 5	flase	7.2.2

9.26 wctp-ResponseHeader

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
responseToMessageID	Mandatory	String	N/A	1 to 32	N/A	8.2.3.1
responseTimestamp	Optional	DateTime	N.A	19 to 23	N/A	7.4
respondingToTimestamp	Optional	DateTime	N/A	19 to 23	N/A	7.4
onBehalfOfRecipientID	Optional	Address	N.A	1 to 128	N/A	8.2.3.1

9.27 wctp-Notification

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
type	Mandatory	Notification	N/A	4 to 9	N/A	7.5

9.28 wctp-Failure

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
(PCDATA)	Optional	String	N/A	1 to 254	N/A	7.3.2, 7.5
errorCode	Mandatory	PositiveNumber	1 to 9999	1 to 4	N/A	7.3.2, 7.5
errorText	Optional	String	N/A	1 to 128	N/A	7.3.2, 7.5

9.29 wctp-Success

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
(PCDATA)	Optional	String	N/A	1 to 254	N/A	7.3.1
successCode	Mandatory	PositiveNumber	1 to 9999	1 to 4	N/A	7.3.1
successText	Optional	String	N/A	1 to 128	N/A	7.3.1

9.30 wctp-PollForMessages

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
pollerID	Mandatory	PollerID	N/A	1 to 128	N/A	4.1.3, 8.6.2
securityCode	Optional	PassCode	N/A	1 to 16	N/A	8.6.2.1
maxMessagesInBatch	Optional	NonNegative Number	0 to 99	1 to 2	N/A	8.6.2.1

9.31 wctp-PollResponse

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
minNextPollInterval	Optional	TimeInterval	0 to 3600	1 to 4	N/A	8.6.2.1.1

9.32 wctp-MessageReceived

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
sequenceNo	Mandatory	String	N/A	1 to 32	N/A	8.6.2.1.1

9.33 wctp-Message

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
sequenceNo	Mandatory	String	N/A	1 to 32	N/A	8.6.2.2

9.34 wctp-SubmitClientHeader

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
submitTimestamp	Mandatory	DateTime	N/A	19 to 23	N/A	5.1.3, 7.1.3

9.35 wctp-ClientOriginator

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
senderID	Mandatory	Address	N/A	1 to 128	N/A	9.2.1.1
miscInfo	Optional	String	N/A	1 to 16	N/A	9.2.1.1

9.36 wctp-ClientMessageControl

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
sendResponsesToID	Mandatory	Address	N/A	1 to 128	N/A	7.1.4.1
allowResponse	Optional	TrueFalse	N/A	4 to 5	true	7.1.4.2
notifyWhenQueued	Optional	TrueFalse	N/A	4 to 5	false	7.1.4.3
notifyWhenDelivered	Optional	TrueFalse	N/A	4 to 5	false	7.1.4.4
notifyWhenRead	Optional	TrueFalse	N/A	4 to 5	false	7.1.4.5
deliveryPriority	Optional	DeliveryPriority	N/A	3 to 6	NORMAL	7.1.4.6
deliveryBefore	Optional	DateTime	N/A	19 to 23	N/A	7.1.4.7
deliveryAfter	Optional	DateTime	N/A	19 to 23	N/A	7.1.4.8
preformatted	Optional	DateTime	N/A	4 to 5	false	7.1.4.9
allowTruncation	Optional	DateTime	N/A	4 to 5	true	7.1.4.10

9.37 wctp-ClientSuccess

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
(PCDATA)	Optional	String	N/A	1 to 254	N/A	7.3.1
successCode	Mandatory	PositiveNumber	1 to 9999	1 to 4	N/A	7.3.1
successText	Optional	String	N/A	1 to 128	N/A	7.3.1
trackingNumber	Mandatory	String	N/A	1 to 16	N/A	9.2.3.1

9.38 wctp-ClientQuery

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
senderID	Mandatory	Address	N/A	1 to 128	N/A	9.3
recipientID	Mandatory	Address	N/A	1 to 128	N/A	9.3
trackingNumber	Mandatory	String	N/A	1 to 16	N/A	9.3

9.39 wctp-ClientQueryResponse

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
minNextPollInterval	Optional	String	0 to 3600	1 to 4	N/A	9.3

9.40 wctp-ClientMessageReply

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
MCRReplyChoice	Optional	TrueFalse	N/A	4 to 5	false	7.2.2

9.41 wctp-ClientResponseHeader

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
`responseTimestamp`	Optional	DateTime	N/A	19 to 23	N/A	7.4
`respondingToTimestamp`	Optional	DateTime	N/A	19 to 23	N/A	7.4

9.42 wctp-VersionQuery

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
`inquirer`	Mandatory	String	N/A	1 to 128	N/A	6.1
`dateTime`	Optional	DateTime	N/A	19 to 23	N/A	6.1

9.43 wctp-VersionResponse

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
`responder`	Mandatory	TransportAddress	N/A	3 to 128	N/A	6.2
`dateTimeOfRep`	Optional	DateTime	N/A	19 to 23	N/A	6.2
`inquirer`	Conditional[2]	String	N/A	1 to 128	N/A	6.2
`dateTimeOfReq`	Conditional[3]	DateTime	N/A	19 to 23	N/A	6.2
`invalidAfter`	Optional	DateTime	N/A	19 to 23	N/A	6.2

9.44 wctp-ContactInfo

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
`email`	Optional	EmailAddress	N/A	5 to 128	N/A	6.2.1
`phone`	Optional	PhoneNumber	N/A	10 to 32	N/A	6.2.1
`www`	Optional	WWWAddress	N/A	3 to 128	N/A	6.2.1
`info`	Optional	String	N/A	1 to 255	N/A	6.2.1

[2] Only returned if provided in original request.

[3] Only returned if provided in original request.

9.45 wctp-DTDSupport

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
dtdName	Mandatory	Version	N/A	8 to 24	N/A	6.2.3
verToken	Optional	String	N.A	1 to 16	N/A	6.2.3
supportType	Optional	SupportType	N/A	9 to 12	Supported	6.2.3
exceptions	Optional	YesNo	N/A	2 to 3	N/A	6.2.3
supportUntil	Optional	DateTime	N/A	19 to 23	N/A	6.2.3
replacement	Optional	Version	N/A	8 to 24	N/A	6.2.3

9.46 wctp-LookupMessageControl

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
messageID	Mandatory	String	N/A	1 to 32	N/A	8.3.1
transactionID	Optional	String	N/A	1 to 32	N/A	8.3.1
sendResponsesToID	Optional	Address	N/A	1 to 128	N/A	8.3.1

9.47 wctp-LookupResponse

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
responseToMessageID	Mandatory	String	N/A	1 to 32	N/A	8.5.2
transactionID	Optional	String	N/A	1 to 32		8.2.1.1

9.48 wctp-LookupData

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
maxMessageLength	Mandatory	NonNegative Number	0 to 10,485,760	1 to 8	N/A	8.5.2.1
mcrSupported	Optional	TrueFalse	N/A	4 to 5	N/A	8.5.2.1
canRespond	Optional	TrueFalse	N/A	4 to 5	N/A	8.5.2.1

9.49 wctp-DeviceLocationControl

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
messageID	Mandatory	String	N/A	1 to 32	N/A	8.4.1
transactionID	Optional	String	N/A	1 to 32	N/A	8.4.1
sendResponsesToID	Optional	Address	N/A	1 to 128	N/A	8.4.1
fixType	Optional	String	N/A	7 to 11	“estimated”	8.4.1

9.50 wctp-DeviceLocationResponse

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
responseToMessageID	Mandatory	String	N/A	1 to 32	N/A	8.5.3
transactionID	Optional	String	N/A	1 to 32	N/A	8.5.3

9.51 wctp-LocationData

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
status	Mandatory	String	N/A	2 to 9	N/A	8.5.3.1
fixType	Mandatory	String	N/A	7 to 11	“estimated”	8.5.3.1
fixTimeStamp	Optional	DateTime	N/A	19 to 23	N/A	8.5.3.1
longitude	Optional	Location	N/A	11	N/A	8.5.3.1
latitude	Optional	Location	N/A	11	N/A	8.5.3.1
altitude	Optional	Number	-32768 to +32767 meters	1 to 6	N/A	8.5.3.1

9.52 wctp-ReturnToSvc

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
(PCDATA)	Optional	Address	N/A	1 to 128	N/A	8.6.1.1

9.53 wctp-MsgMultiHeader

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
submitTimestamp	Mandatory	DateTime	N/A	19 to 23	N/A	5.1.3, 7.1.3

9.54 wctp-MsgMultiControl

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
messageID	Mandatory	String	N/A	1 to 32	N/A	8.2.1.1
transactionID	Optional	String	N/A	1 to 32	N/A	8.2.1.1
sendResponsesToID	Optional	Address	N/A	1 to 128	N/A	7.1.4.1
allowResponse	Optional	TrueFalse	N/A	4 to 5	true	7.1.4.2
notifyWhenQueued	Optional	TrueFalse	N/A	4 to 5	false	7.1.4.3
notifyWhenDelivered	Optional	TrueFalse	N/A	4 to 5	false	7.1.4.4
notifyWhenRead	Optional	TrueFalse	N/A	4 to 5	false	7.1.4.5
deliveryPriority	Optional	DeliveryPriority	N/A	3 to 6	NORMAL	7.1.4.6
deliveryBefore	Optional	DateTime	N/A	19 to 23	N/A	7.1.4.7
deliveryAfter	Optional	DateTime	N/A	19 to 23	N/A	7.1.4.8
preformatted	Optional	DateTime	N/A	4 to 5	false	7.1.4.9
allowTruncation	Optional	DateTime	N/A	4 to 5	true	7.1.4.10
allRecipsRequired	Optional	TrueFalse	N/A	4 to 5	true	8.2.2.2

9.55 wctp-SendMsgMultiResponse

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
maxNumRecips	Mandatory	PositiveNumber	1 to 9999	1 to 4	N/A	8.2.2.4
numValidRecips	Optional	NonNegative Number	0 to 9999	1 to 4	N/A	8.2.2.4
numInvalidRecips	Optional	NonNegative Number	0 to 9999	1 to 4	N/A	8.2.2.4

9.56 wctp-FailedRecipient

Attribute Name	Disposition	Type	Range	Length	Default	Ref.
recipientId	Mandatory	Address	N/A	1 to 128	N/A	8.2.2.5
errorCode	Mandatory	PositiveNumber	1 to 9999	1 to 4	N/A	8.2.2.5
errorText	Optional	String	N/A	1 to 128	N/A	8.2.2.5

Appendix E Error and Success Codes

Code	Definition	Comments
200 Series Success Codes		
200	Acknowledged	The operation has been received and a first level of validation has been performed. Further validation may result in additional responses. Other responses to this message, such as delivery notifications, may occur as per the operations requested.
210	Deprecated version	The WCTP version used has been deprecated.
211	Message exceeds allowable length	Message exceeds allowable message length. Message will be delivered, but truncated at max.
212	Delivery ACK Not Supported by Device	The Device does not support Delivery Acknowledgements. Message will be delivered but no DELIVERED notification will be returned.
213	Read ACK Not Supported by Device	The Device does not support Read Acknowledgements. Message will be delivered but no READ notification will be returned.
215	Multiple Choice Response Not Supported by Device	Device does not support Multiple Choice Responses. Converted to appended text.
216	Multiple Choice Responses exceeds allowable length	Multiple Choice Response text is too long. Truncated to max
217	Max Multiple Choice Responses exceeded	Message contains too many Multiple Choice Responses. Truncated at max
218	Delivery ACK Not Supported	The Network does not support Delivery Acknowledgements. Message will be delivered, but no DELIVERED notification will be returned.
219	Read ACK Not Supported	The Network does not support Read Acknowledgements. Message will be delivered but no READ notification will be returned.
220	Multiple Choice Responses Not Supported	The Network does not support Multiple Choice Responses. Converted to appended text.

300 Series Protocol Violation Error Codes		
300	Operation not supported	The requested WCTP operation is not supported by this system.
301	Input can not be parsed	Input message is garbled. Invalid XML format. Could be caused by binary data or non-XML data being sent.
302	XML validation error	Input is not a well formed XML document and cannot be processed properly.
303	Version error	The version of the input data is not supported.
304	Cannot accept operation	The requested operation can not be performed until messages are pulled using the PollForMessages operation.

400 Series Content Error Codes		
400	Function not supported	The requested function, is not supported for this operation.
401	Invalid SenderID	The senderID is invalid.
402	Invalid security code	The security code for this senderID is invalid.
403	Invalid recipientID	The recipientID is invalid.
404	Invalid authorization code	The authorization code for this recipientID is invalid.
405	Invalid Date or Time Format	A date and/or time value is not in the WCTP Standard date and/or time format.
406	Date/Time Ranges specify an unsupported combination	A combination of a request to send a message after a specific time but before another time is incorrectly specified. The 'before' date/time value must be beyond the 'after' date/time value.
407	Invalid date/time period	The specified date and/or time is beyond the time range supported by the system.
408	Date/time specified has	The date/time specified has already passed and the operation requested can

Code	Definition	Comments
	already expired	not be performed as specified.
411	Message exceeds allowable length	Message exceeds allowable message length.
412	Delivery ACK Not Supported by Device	The Device does not support Delivery Acknowledgements.
413	Read ACK Not Supported by Device	The Device does not support Read Acknowledgements.
414	Multiple Choice Response Not Supported by Device	Device does not support Multiple Choice Responses.
415	Multiple Choice Responses exceeds allowable length	Multiple Choice Response text is too long.
416	Max Multiple Choice Responses exceeded	Message contains too many Multiple Choice Responses.
417	Delivery ACK Not Supported	The Network does not support Delivery Acknowledgements.
418	Read ACK Not Supported	The Network does not support Read Acknowledgements.
419	Multiple Choice Responses Not Supported	The Network does not support Multiple Choice Responses.
420	Invalid address/alias	The alias is not known to the carrier or the address does not conform to an address format
430	Binary Message Not Supported	Device does not support Binary Messages.
431	Subscriber Limit Exceeded	Message limit to subscriber has temporarily been exceeded
432	Invalid Poller or Security Code	Invalid combination of Poller ID with Security Code

<table>
<tr><th colspan="3">500 Series Message Processing Error Codes (Permanent)</th></tr>
<tr><td>500</td><td>Message has timed out</td><td>The message could not be delivered within a network specific maximum delivery time period.</td></tr>
<tr><td>501</td><td>Message has expired</td><td>The message could not be delivered within the delivery time period specified when it was submitted.</td></tr>
<tr><td>502</td><td>No further responses possible</td><td>The specified message ID will no longer be able to receive responses because of network or device limitations.</td></tr>
<tr><td>503</td><td>Sender permanently disabled due to excessive polling or traffic</td><td>Manual intervention by the operator of the WCTP server is required. The system performing polling has ignored warnings to reduce the polling rate or warnings that traffic levels are over the allowed quota. Service has been barred.</td></tr>
<tr><td>504</td><td>Unknown Message Reference</td><td>Message reference (a combination of senderID, recipientID and a trackingNumber) presented in ClientQuery is not recognized.</td></tr>
</table>

<table>
<tr><th colspan="3">600 Series Message Processing Error Codes (Temporary)</th></tr>
<tr><td>600</td><td>Poll rate exceeded</td><td>Warning. Polling for messages and status is occurring too frequently. The error text, which accompanies this status, may provide a value (in seconds) for the maximum allowed polling rate.</td></tr>
<tr><td>601</td><td>Excessive polling</td><td>Warning. Polling is occurring at a rate beyond the allowed limitations. If this continues, the polling system will be permanently disabled (see error 503).</td></tr>
<tr><td>602</td><td>Traffic rate exceeded</td><td>Warning. The rate at which messages are entering the system exceeds the allowed quota. Slow down the message entry rate.</td></tr>
<tr><td>603</td><td>Traffic rate excessive</td><td>Warning. The rate at which traffic is entering the system exceeds the allowed quota. If this continues the sender will be permanently disabled (see error 503).</td></tr>
</table>

Code	Definition	Comments
604	Internal Server Error	The WCTP gateway encountered an unexpected condition that prevents it from processing the message.
605	Service Unavailable	The WCTP gateway is unable to handle the request due to either temporary overloading or server maintenance.

700 Series Experimental Error Codes		
700	Maximum Number of Requests Exceeded	The session has been terminated. The maximum number of requests for a single connection has been exceeded.
701	Excessive Rate for Requests	The session has been terminated. The maximum rate for requests per unit time for a single connection has been exceeded.
702	Maximum Time Limit Exceeded	The session has been terminated. The maximum length of time allowed for a single connection has been exceeded.

900 Series Experimental Error Codes		
900-999	Implementation specific	Implementations may experiment with new codes and new features employing these codes, without fear of interfering with future versions of the WCTP specification, by utilizing code values within this range of numbers. The 900 series codes will never be specified in an officially released WCTP specification. Presumably, once an experimental implementation is deemed to be operational, the developer of this new capability may submit a request to the appropriate WCTP Standards Committee to request that an official status or error code designation be allocated.

Appendix F WCTP Committee Acknowledgement

The following committee members contributed to the development, enhancements, and promotion of this specification.

Chairman

Gagan Puranik	SkyTel	gagan.puranik@mci.com

Past Chairman

Allan Angus	WebLink Wireless	Allan.Angus@weblinkwirelss.com
Edward “Ted” Sumner	Arch Wireless	Edward.Sumner@arch.com
Demitirius Nelon	Arch Wireless	Demitrius.Nelon@arch.com
Tim Dockins	WebLink Wireless	Tim.Dockins@weblinkwireless.com

Originating Authors

Jay Moskowitz	RTS Wireless	jmosk@rtswireless.com
Dr. Yuri Salkinder	RTS Wireless	yuri.salkinder@rtswireless.com

Current Drafting Committee Members

Committee Member	Company
Tereus Scott	GTES
Pat Adams	USA Mobility
James Dabbs	TGA
Allan Angus	Trace Technologies
Icarus Chau	GTES
Ron Mercer	Global Fax Networks
Brian Claise	TGA
Scott Humphries	SkyTel
Michael Almond	Space Data Corporation
Paul Reeder	Telemisphere
Bonnie Shapbell	SkyTel
Laura Halliday	GTES
Doug Potts	MonComm

Past Drafting Committee Members

Committee Member	Company
Dr. Alfred Adler	Motorola
Raj Cherukuri	SkyTel
Dr. Michael Christiansen	Pagenet
Vic Cox	WebLink Wireless
John Davis	WebLink Wireless
Bruce Deer	SkyTel
Tim Dockins	WebLink Wireless
Don Gayton	Glenayre
Mike Johnson	SkyTel
Jerry Karasz	SkyTel
Anthony Klinkert	Pagenet
Brian Laird	Glenayre
Kris Latham	Arch Wireless
Kyung Tae Mun	WebLink Wireless
Jay Moskowitz	RTS Wireless
Demitrius Nelon	Arch Wireless
Bijan Nowroozi	Pagenet
Garland Phillips	Motorola
Jeff Poe	Space Data Corporation
Dr. Yuri Salkinder	RTS Wireless
Karl Schlieber	Bell South Wireless Data
Bill Scott	Pagenet
Mehrshad Setayesh	WebLink Wireless
Dwight Smith	Motorola
Ted Sumner	Pagenet
Benjamin Sun	Pagenet
Chris Wells	Glenayre
Shawn Welsh	SkyTel
Wiley Wilkins	Pagenet
Craig Green	USA Mobility
John de Boer	USA Mobility

Appendix G Revision History

Document Number	Date	Description
wctpv1r3 update1	May 18, 2006	Added new section 2.3 to provide clarification to WCTP V1.3 regarding other secure transport mechanisms such as HTTPS and VPN.
wctpv1r3	April 22, 2005	Updated by Tereus Scott Added the Device Location operations: wctp-DeviceLocation and wctp-DeviceLocationResponse. Fixed typos in 3.3.4, 3.5, 8.5, 8.5.2, 9.2.3, 9.3, B.3.7, B.3.8, B.3.11 Fixed bad cross reference in section 3.3.3, 7.3 & 8.2.3 Cleaned up rev 1.2 change history (accepted all changes) Added FLEXsuite as a option to the type attribute of wctp-TransparentData on page A-11 Moved Table 2 into section 3.3.2 where it belongs.
wctpv1r2	May 12, 2003	Incorporated Technical notes from WCTP 1.1 Added MCR enhancements, multiple recipient support and return to service operation Reorganized document to clarify the protocol for implementers
wctpv1r1.doc	December 29, 2000	Added wctp-VersionQuery. Removed wctp-BindDomainAlias. Addressing clarified and corrigenda incorporated
wctpv1r0.doc	June 6, 2000	Updated specification to reflect WCTP as a PCIA supported protocol. Update references to the WCTP DTD to show that it now exists at the PCIA Web site.
wctpv0r7.doc	November 1, 1999	Final revision of the first “official” version of the protocol presented to the PCIA. Accepted as a standard by the Paging Technical Committee of the Personal Communication Industry Association (PCIA). Use of the uuencode scheme for the transparent data transfer is discontinued, and the use of the

		XML “canonical” encoding is discouraged in favor of base64 encoding.
wctpv0r6.doc	June 18, 1999	Minor typos and inconsistencies are fixed in the use cases. All the use cases were validated through an XML parser.
wctpv0r5.doc	May 17, 1999	Release of drafting committee preliminary version 0.4 with a major portion of the 21 defined use cases included.
wctpv0r4.doc	April 9, 1999	Incorporated changes agreed to by the members of the drafting committee, developed during a series of meetings and conference calls. This is major update to the DTD and rewrite of the specification from the previous release. Changes span the entire document and are too numerous to list individually.
wctpv0r3.doc	February 22, 1999	Re-release of specification under the name of the Messaging Standards Committee instead of RTS Wireless, the original author.
wctpv0r2.doc	January 29, 1999	Initial Release by Jay Moskowitz, CTO, RTS Wireless, a.k.a. Real Time Strategies, Inc. Introductory specification released after a presentation to the Messaging Standards Committee, January 13, 1999. As a result of that meeting, certain aspects of WCTP have been eliminated from this specification so as to simplify its description and to avoid the addressing of implementation details which should be outside of the scope of this document.