WCTP Developers Guide
Arch Wireless Implementation

Table of Contents

1 About Arch Wireless 1
2 Introduction to WCTP 3
3 Naming Conventions Used in this Manual 4
4 Common Mistakes in HTTP and XML with WCTP 5
5 WCTP Operations 6
5.1 Standard WCTP Operations 6
5.1.1 Submit Client Message 6
5.1.2 Client Query 7
5.2 Enterprise WCTP Operations 9
5.2.1 Submit Request 9
5.2.2 Poll For Messages 11
5.2.3 Status Info 11
5.3 Versioning 12
5.3.1 Tokens 12
5.3.2 Version Query 12
6 Messaging Conventions 14
6.1 Messaging to Devices 14
6.2 Messaging to Applications 14
6.3 Messaging to Applications from a Device 14
7 Examples of Common Applications 15
7.1 Client Sending a Message 15
7.2 Client Getting a Delivery Confirmation 17
7.3 Enterprise Client Sending a MCR Message 19
7.4 Device Originated Message to an Application 22
8 Example Enterprise Application Framework 26
8.1 Proxy Model 26
8.2 Push Based Model 27
8.3 Central Database for the Proxy 27
8.3.1 Inserting a Receipt 28
8.3.2 Querying a Receipt 28
8.3.3 Updating a Receipt 28
8.3.4 Purging the Database 28
8.4 Using a Proxy in the Enterprise 28
8.4.1 Distributed Applications 28
8.4.2 Access Control 28
8.4.3 Logging 29
9 Where to Find the Protocol 30
10 Commercial Software 31
11 Available API's 32
12 What's included on the CD? 33

1 About Arch Wireless

Arch Wireless™ is Leading the Way in Wireless Communications
Arch Wireless Inc. is the premier provider of end-to-end wireless enterprise solutions, wireless e-mail, instant text messaging and mobile Internet access. We offer a broad range of wireless data solutions to companies looking to gain a competitive advantage by enhancing productivity as well as to individuals who want to make their daily on-the-go lives easier and more manageable.

Arch Wireless has more than 250 offices and company stores across the U.S., serving millions of individuals and hundreds of Fortune 1000 and smaller companies. Our expert consultants are ready to work directly with businesses of all sizes to help create and integrate a custom wireless data solution into their existing infrastructure.

All of Arch Wireless' innovative, interactive solutions take advantage of the powerful Arch Wireless data network—the most robust and reliable of its kind in North America. With superior coverage throughout all 50 states, D.C., Canada, the Caribbean, Mexico and Puerto Rico, Arch customers get the information that's needed most—from virtually anywhere business or life takes them.

The Arch Wireless Network Delivers
The Arch Wireless Network uses the highly efficient ReFLEX® 25 digital point-to-multipoint platform and offers higher signal strength, in-building penetration, and far greater landmass and population coverage of cellular, PCS, and wireless broadband combined. Today the Arch Network can provide coverage to more than 93% of the U.S. population. With ongoing enhancements, our network will be providing subscribers with superior two-way wireless data services for years to come.

The Benefits of Arch Wireless Are Clear

- **Ubiquity** - Arch's network provides unmatched coverage in all 50 states, DC, Canada, Mexico, and the Caribbean.
- **Reliability** - The strength of the signal delivered over the Arch Wireless network assures a higher quality transmission and substantially greater assurance of message receipt—indoors and out.
- **Bandwidth** – Arch has extensive spectrum assets, positioning us to effectively handle the growth and capacity in two-way wireless communication.
- **Customization** - To create the perfect wireless data solution tailored to your specific needs, Arch Wireless' expert consultants work directly with your organization every step of the way.

?? **Integration** - Arch Wireless uses a open standard connection protocols, making our services easy to integrate and expand upon as your needs grow.

Our Management Team

C. Edward Baker, Jr., Chairman and Chief Executive Officer
Lyndon R. Daniels, President and Chief Operating Officer
Steven C. Gross, Executive Vice President, Sales and Marketing
Paul H. Kuzia, Executive Vice President, Technology and Regulatory Affairs
J. Roy Pottle, Executive Vice President and Chief Financial Officer
John B. Saynor, Executive Vice President, Corporate Development
Peter Barnett, Senior Vice President, Operations, and Chief Information Officer
Patricia A. Gray, Senior Vice President and General Counsel
Christopher J. Madden, Senior Vice President, Human Resources
Mark L. Witsaman, Senior Vice President, Technology, and Chief Technology Officer
Gerald J. Cimmino, Vice President and Treasurer
George W. Hale, Vice President, Finance and Controller
Robert W. Lougee, Jr., Vice President, Corporate Communications and Investor Relations

Our Corporate Headquarters

1800 West Park Drive, Suite 250
Westborough, MA, 01581
Phone: 508-870-6700
Fax: 508-836-3626

2 Introduction to WCTP

WCTP (Wireless Communications Transport Protocol) is based on XML 1.0 and is currently maintained by the Personal Communications Industry Association (PCIA.) WCTP was originally a joint project between a wireless carrier, some manufacturers, and an ASP. WCTP was given to the PCIA as a proposal for a two-way protocol to support the newly deployed networks. All of the member carriers of PCIA adopted WCTP as the official two-way wireless messaging protocol. The existing version of the protocol was renamed 1.0 and adopted. After a couple of carriers implemented the protocol a working group was formed to enhance WCTP. WCTP 1.1 was recently released, and 1.2 and 2.0 versions are currently under development. In the 1.1 version of the protocol versioning was introduced. It is not efficient to query version information every time a connection to a server is made. A mechanism was introduced to allow a token to exist in the response document for every transaction. This token could then be compared with the previously stored token to determine if any changes had taken place with the current implementation. If change had occurred, then a Version Query would be appropriate. One of the additional requested feature sets for 2.0 is SMS functionality. The 1.2 version will clear up inconsistencies in the protocol and improve the readability of the document.

WCTP is not designed for a given wireless protocol, and bi-directional communications are supported. The transport model for the protocol is request/response oriented. Every WCTP document that is delivered will receive a WCTP document as its reply. This could either be a success or failure notification.

The protocol is device and platform independent. In fact, many devices do not have any native support for WCTP. Currently, the WCTP protocol is used to access are either pagers or PDA's. These could be anything from numeric one-way to two-way pagers and PDA's with wireless connectivity. WCTP has already been used to deliver messages to devices that operate on Flex, ReFlex, and Mobitex. There is also a SMS gateway that has the capability to translate WCTP requests.

3 Naming Conventions Used in this Manual

WCTP Server	The term WCTP server is used to identify a WCTP gateway at the carrier. In other words, this is the final destination for the WCTP communication before it reaches the device.
Receipts	A receipt is a confirmation of delivery.
Transient Client	A transient client talks to a WCTP server but does not have a permanent 'home.' A transient client may later go back to check for receipts and device replies using a Client Query, but a device is not able to reply to a transient client message. A transient client also can't have receipts posted back to it. A transient client uses the Submit Client Message operation to send WCTP messages.
Enterprise Client	An enterprise client picks up where a transient client leaves off. A device can reply to messages sent by an enterprise client. An enterprise client can also specify an application for replies and receipts to be posted to.
Enterprise Application	An enterprise application is an application that lives on the internet that a WCTP server posts operations to. The WCTP server may post receipts or replies to the enterprise application.
Device ID	A device ID is the number that is assigned to the device by your carrier.

4 Common Mistakes in HTTP and XML with WCTP

WCTP currently has only one implemented transport model. This is based on HTTP. HTTP has several headers that are important to the functionality of WCTP. The most common area of mistake is the Content-Length field. If you do not have the correct value in this field, it is likely that you will receive a WCTP 301 Parse Error. An assumption is also made about the nature of HTTP 1.1. No reference implementation currently exists to allow transaction chaining via HTTP 1.1. If HTTP 1.1 is used, you should expect to see the Connection-Closed directive after receiving a response from the remote WCTP server. Problems have also been discovered in using the same IP address for multiple web sites. The naming convention of http://wctp.xxx.xxx/wctp is intended to allow for shorter addresses with over the air protocols. You may need to diagnose the configuration of your HTTP server to ensure that no problems arise from this convention.

Other modes of transport are possible, but have not been tested, and not all operations in WCTP lend themselves to asynchronous transfer. It is expected that at some point in the future a raw port will be assigned for WCTP communications that does not require the construction and destruction of an HTTP connection.

Another common problem arises from the nature of CDATA in XML. It is necessary to escape XML entities:

>	>
<	<
&	&
”	"
‘	'

If you want a message to display one of these characters on the device, you need to replace them with their escaped values.

WCTP is an example of XML, and a DTD is the current document used to ensure that WCTP is not only well formed, but valid. The URI for the DTD you are using must be one that can be resolved by both participants in WCTP communications. Some parsers have also been found to attempt to request a DTD from a remote source even when this feature is disabled.

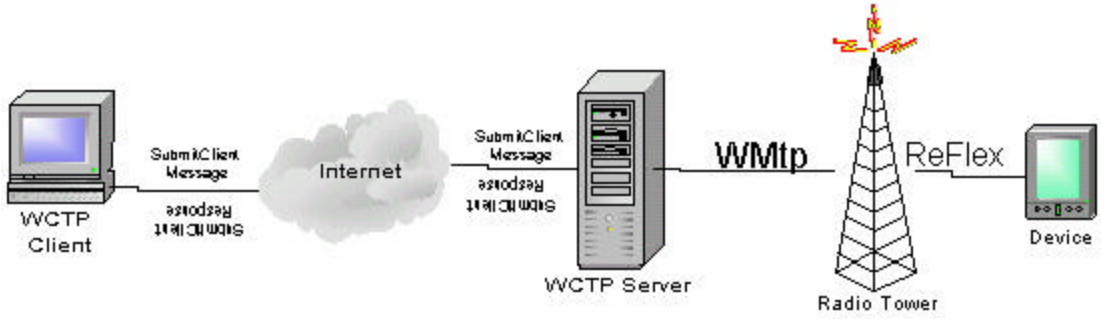
WCTP gains most of its power and simplicity by leveraging the existing standards of HTTP and XML. This does not provide complete functional abstraction for all implementations, so be sure that you are truly familiar with HTTP and XML before assuming that the remote system you are communicating with is flawed. At least, be sure that you have a copy of the current RFC/Specification from http://www.ietf.org/ or http://www.w3.org/.

5 WCTP Operations

For the purpose of this manual, all WCTP operations have been divided into two categories. Standard operations are used by transient clients. Enterprise operations are used by both enterprise clients and enterprise applications. Versioning is a special kind of operation that is explained later in this chapter.

5.1 Standard WCTP Operations

5.1.1 Submit Client Message



A WCTP Submit Client Message is the most basic form of a WCTP operation. A transient client may obtain receipts and device replies using the Client Query operation. A transient client may not route replies to another device. A transient client formats a packet and submits it to a WCTP server. The server checks for any immediate error conditions (such as bad XML or an invalid subscriber) and responds with either a success or failure packet. Required parameters include a sender ID and a recipient ID. Frequently used optional parameters include setting a message as preformatted, including a timestamp, and requesting a confirmation of delivery. The following are some brief definitions of commonly used parameters in a WCTP Client Message:

senderID	Required field used to identify the sender. This is not a well-defined field for transient clients and is expected to be deprecated in the future.
recipientID	Used to identify the device ID that you are sending the message to.
notifyWhenQueued	This lets the server know that you will want to know when the message has been queued to be sent to the device.
notifyWhenDelivered	This lets the server know that you will want to know when the message has been sent to the device.
preformatted	This lets the server know to preserve white space and carriage returns in the payload. The user should be made aware that the white space and carriage returns may be counted as billed characters.
submitTimestamp	The timestamp indicates the time that the message was sent in GMT.

The following XML is a WCTP Submit Client Message packet:

```
<!DOCTYPE wctp-Operation SYSTEM "http://dtd.wctp.org/wctp-dtd-v1r1.dtd">
<wctp-Operation wctpVersion="wctp-dtd-v1r1">
 <wctp-SubmitClientMessage>
  <wctp-SubmitClientHeader submitTimestamp="2001-07-31T17:56:05">
   <wctp-ClientOriginator senderID="sender@arch.com"/>
   <wctp-ClientMessageControl
    notifyWhenQueued="true"
    notifyWhenDelivered="true"
    notifyWhenRead="false"
   />
   <wctp-Recipient recipientID="DEVICEID@arch.com"  />
  </wctp-SubmitClientHeader>
  <wctp-Payload>
   <wctp-Alphanumeric>test</wctp-Alphanumeric>
  </wctp-Payload>
 </wctp-SubmitClientMessage>
</wctp-Operation>
```

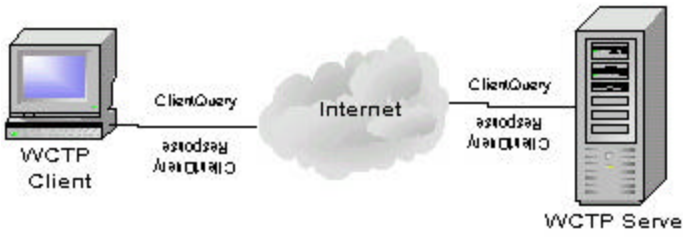
The following XML is a response from a WCTP server for a valid message.

```
<?xml version="1.0"?>
<!DOCTYPE wctp-Operation SYSTEM "http://dtd.wctp.org/wctp-dtd-v1r1.dtd">
<wctp-Operation wctpVersion="wctp-dtd-v1r1" wctpToken="11AA">
 <wctp-SubmitClientResponse>
  <wctp-ClientSuccess
  successCode="200"
  successText="Accepted"
  trackingNumber="0004997072"
  >

      Your message for DEVICEID@arch.com has been accepted for delivery.

  </wctp-ClientSuccess>
 </wctp-SubmitClientResponse>
</wctp-Operation>
```

5.1.2 Client Query



A Client Query is sent by a transient client to a WCTP server. The client must first send a WCTP Client Message that specifies notification upon delivery. The server will

respond with a tracking number. The tracking number is then used to obtain delivery status (either QUEUED or DELIVERED) and device replies. The required parameters for a WCTP Client Query are a sender ID, recipient ID, and tracking number. The following table describes these fields in more detail:

senderID	The senderID that was indicated in the original message. Typically, this field is used by a WCTP server to key on a database.
recipientID	The recipientID that is indicated in the original message. Typically, this field is used by a WCTP server to key on a database.
TrackingNumber	The tracking number that was returned by the WCTP server when the message was originally sent.

The following XML shows a Client Query operation:

```
<?xml version="1.0"?>
<!DOCTYPE wctp-Operation SYSTEM "http://dtd.wctp.org/wctp-dtd-v1r1.dtd">
<wctp-Operation wctpVersion="wctp-dtd-v1r1">
 <wctp-ClientQuery
  senderID="sender@arch.com"
  recipientID="DEVICEID@arch.com "
  trackingNumber="0004997072"
 />
</wctp-Operation>
```

The following shows the XML response from a WCTP server reporting the message is QUEUED, but not DELIVERED:

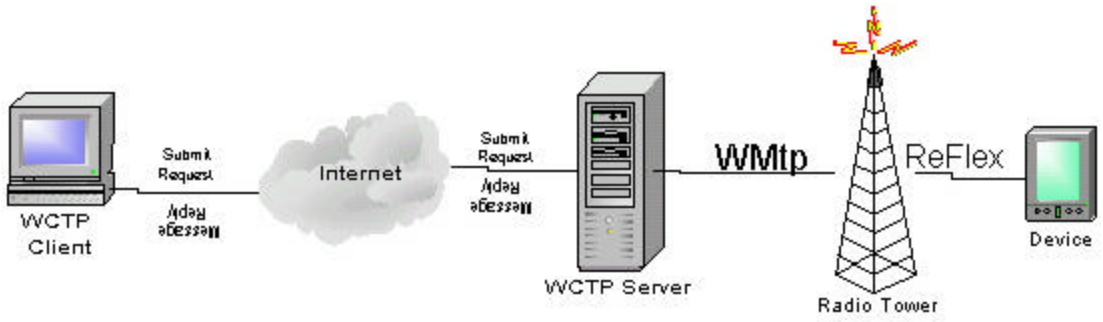
```
<?xml version="1.0"?>
<!DOCTYPE wctp-Operation SYSTEM
"http://www.pcia.com/wireres/protocol/dtd/wctpv1-0.dtd">
<wctp-Operation wctpVersion="wctp-dtd-v1r1" wctpToken="11AA">
 <wctp-ClientQueryResponse>
  <wctp-ClientMessage>
   <wctp-ClientStatusInfo>
    <wctp-ClientResponseHeader>
     <wctp-Originator senderID="sender@arch.com " />
     <wctp-Recipient recipientID="DEVICEID@arch.com " />
    </wctp-ClientResponseHeader>
    <wctp-Notification type="QUEUED" />
   </wctp-ClientStatusInfo>
  </wctp-ClientMessage>
 </wctp-ClientQueryResponse>
</wctp-Operation>
```

The following XML shows the response from a WCTP server reporting the message is DELIVERED:

```
<?xml version="1.0"?>
<!DOCTYPE wctp-Operation SYSTEM
"http://www.pcia.com/wireres/protocol/dtd/wctpv1-0.dtd">
<wctp-Operation wctpVersion="wctp-dtd-v1r1" wctpToken="11AA">
 <wctp-ClientQueryResponse>
  <wctp-ClientMessage>
   <wctp-ClientStatusInfo>
    <wctp-ClientResponseHeader>
     <wctp-Originator senderID=" sender@arch.com" />
     <wctp-Recipient recipientID="DEVICEID@arch.com " />
    </wctp-ClientResponseHeader>
    <wctp-Notification type="QUEUED" />
   </wctp-ClientStatusInfo>
  </wctp-ClientMessage>
  <wctp-ClientMessage>
   <wctp-ClientStatusInfo>
    <wctp-ClientResponseHeader>
     <wctp-Originator senderID=" sender@arch.com" />
     <wctp-Recipient recipientID="DEVICEID@arch.com " />
    </wctp-ClientResponseHeader>
    <wctp-Notification  type="DELIVERED" />
   </wctp-ClientStatusInfo>
  </wctp-ClientMessage>
 </wctp-ClientQueryResponse>
</wctp-Operation>
```

5.2 Enterprise WCTP Operations

5.2.1 Submit Request



A WCTP Submit Request is an enterprise message that is sent to the gateway. The Submit Request operation offers increased functionality to over the Submit Client Message operaton. Most notably, a Submit Request uses the sender ID field to indicate where a device should respond. Possible destinations include another device or an application. An enterprise client will format a Submit Request operation and submit it to a WCTP server. The server parses the XML for any immediate error conditions such as an invalid subscriber or invalid XML. Either a success or failure is returned to the enterprise client. Required fields for a Submit Request are a sender ID, message ID, and

recipient ID. Frequently used fields include submitTimestamp, sendResponsesToID, notifyWhenQueued, notifyWhenDelivered, and preformatted. The following are some brief definitions of commonly used parameters in a WCTP Submit Request:

senderID	This field indicates where to send responses to. This may be a device ID or the URL of an enterprise application.
messageID	This field should uniquely identify the message.
recipientID	Used to identify the device ID that you are sending the message to.
submitTimestamp	The timestamp indicates the time that the message was sent in GMT.
sendResponsesToID	This field can be used to specify an application where you want any device responses or receipts sent to.
notifyWhenQueued	This lets the server know that you will want to know when the message has been queued.
notifyWhenDelivered	This lets the server know that you will want to know when the message has been delivered to the device.
preformatted	This lets the server know to preserve white space and carriage returns in the payload. The user should be made aware that the white space and carriage returns may be counted as billed characters.

The following XML shows a Submit Request operation:

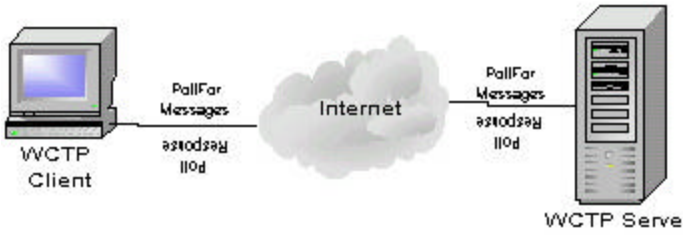
```
<?xml version="1.0"?>
<!DOCTYPE wctp-Operation SYSTEM "http://dtd.wctp.org/wctp-dtd-v1r1.dtd">
<wctp-Operation wctpVersion="wctp-dtd-v1r1">
 <wctp-SubmitRequest>
  <wctp-SubmitHeader
   submitTimestamp="2001-07-31T18:49:06"
  >
   <wctp-Originator
    senderID="MSG:fakeid@fakeserver.com:8080/fakeApplication"
   />
   <wctp-MessageControl
    allowResponse="true"
    messageID="7443axp"
    notifyWhenDelivered="true"
    notifyWhenQueued="true"
    notifyWhenRead="false"
   />
   <wctp-Recipient
    recipientID="DEVICEID@arch.com"
   />
  </wctp-SubmitHeader>
  <wctp-Payload>
```

```
  <wctp-Alphanumeric>test</wctp-Alphanumeric>
 </wctp-Payload>
</wctp-SubmitRequest>
</wctp-Operation>
```

The following XML shows a response from a WCTP server showing a that the message was accepted.

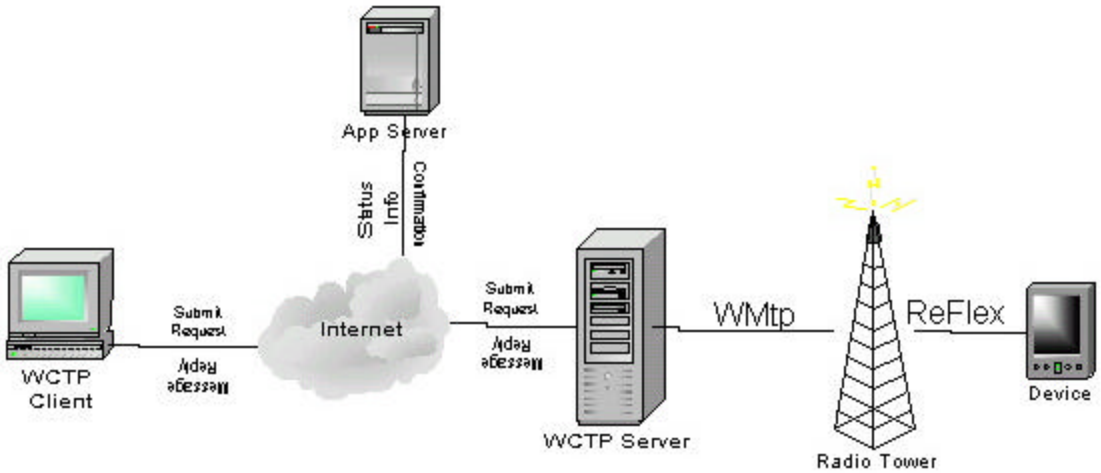
```
<?xml version="1.0"?>
<!DOCTYPE wctp-Operation SYSTEM
"http://www.pcia.com/wireres/protocol/dtd/wctpv1-0.dtd">
<wctp-Operation wctpVersion="wctp-dtd-v1r1" wctpToken="11AA">
 <wctp-Confirmation>
  <wctp-Success successCode="200" successText="Accepted">
       Your message for DEVICEID@arch.com has been accepted for delivery.
  </wctp-Success>
 </wctp-Confirmation>
</wctp-Operation>
```

5.2.2 Poll For Messages



The WCTP Poll For Messages operation is used by an enterprise application to pull receipts or device replies from a WCTP server. However, the preferred method for obtaining receipts is to have the receipts and device responses pushed to an enterprise application.

5.2.3 Status Info



In current implementations, the WCTP Status Info operation is most commonly used by a WCTP server to post receipts and device originated replies to an enterprise application. For example, an enterprise client has indicated that it wants to be notified when a message has been delivered. The enterprise client has indicated to send receipts to an enterprise application at http://someserver:8080/wctpReceipts. At some point, the WCTP server knows that the message has been delivered so it posts a Status Info operation to that URL with the notification type of DELIVERED. Similarly, if an enterprise application indicates to send responses to the previous URL, the WCTP server will post device replies to that address.

5.3 Versioning

The 1.1 revision of WCTP introduced the notion of versioning. Versioning allows a WCTP server to keep standard and enterprise clients informed about the status of the gateway. Two mechanisms are used for versioning in WCTP. Those mechanisms are tokens and the Version Query operation.

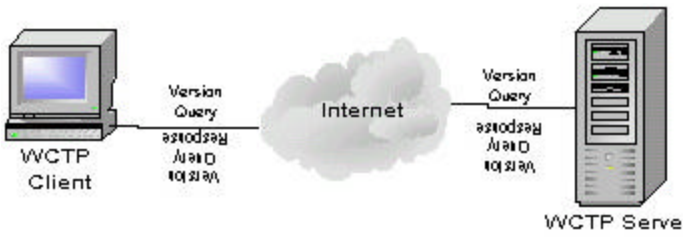
5.3.1 Tokens

A token is used by a WCTP server (or possibly an enterprise application) to indicate to clients that there has been a change in the gateway. While there are not currently any specific conventions to define a token format, any change in the token should be enough for the client to acknowledges changes in the server. This is done using a Version Query operation.

The following XML snippet shows where a version token is located:

<wctp-Operation wctpVersion="wctp-dtd-v1r1" wctpToken="11AA">

5.3.2 Version Query



The Version Query operation is used to acquire information about a WCTP server. The client can acquire the gateway version and a list of DTD's currently supported by the gateway. The required field for a Version Query is the inquirer. To obtain a list of DTD's the listDTDs option should be set to true. The following are some brief definitions of useful fields in the Version Query operation:

inquirer	This field represents a URI that identifies the entity making the version inquiry.
listDTDs	Setting this option to 'true' indicates that you want a list of supported DTDs.

The following XML shows a Version Query:

```
<?xml version="1.0"?>
<!DOCTYPE wctp-Operation SYSTEM "http://dtd.wctp.org/wctp-dtd-v1r1.dtd">
<wctp-Operation wctpVersion=“wctp-dtd-v1r1”>
 <wctp-VersionQuery
  inquirer="sender@arch.com"
  dateTime="2001-07-31T18:53:33"
 />
</wctp-Operation>

<?xml version="1.0"?>
<!DOCTYPE wctp-Operation SYSTEM " http://dtd.wctp.org/wctp-dtd-v1r1.dtd ">
<wctp-Operation wctpVersion=“wctp-dtd-v1r1” wctpToken="11AA">
 <wctp-VersionResponse
  responder="wctp.arch.com/wctp"
  dateTimeOfRsp="2001-07-31T18:54:40"
  inquirer="sender@arch.com"
  dateTimeOfReq="2001-07-31T18:53:33"
 >
  <wctp-DTDSupport
   dtdName=”wctp-dtd-v1r1”
   verToken=”11AA”
   supportType=”Supported”
  />
  <wctp-DTDSupport
   dtdName=”wctp-dtd-v1r0”
   supportType=”Deprecated”
  />
 </wctp-VersionResponse>
</wctp-Operation>
```

6 Messaging Conventions

6.1 Messaging to Devices

To message to a device use the ID that was assigned to the device by your carrier followed by a carrier-specific domain. For example, if your device ID is 1234567, then you use this in the senderID field:

senderID=”1234567@arch.com”

6.2 Messaging to Applications

To message to an enterprise application from a enterprise client you use the URL of your application. In this example, let’s assume that you have an application at http://somewhere:8080/yourApplication/. You would indicate to route responses to an application in the Submit Request operation with the following:

senderID=”MSG:username@somewhere:8080/yourApplication”

Where the format is:

MSG:someid@<server>:port#/application/

If a port is not specified, port 80 is assumed.

Note that “MSG:” is prepended to the URL. This indicates to the server that the senderID is either an address or a poller ID. This manual does not cover polling, so we will assume it is only for enterprise applications at this point.

6.3 Messaging to Applications from a Device

To message to an enterpries application from a device you use the same conventions as listed in section 8.2. The type of response should be set to ‘email’ on the device.

7 Examples of Common Applications

The following applications use the Arch Wireless WCTP Factory API for basic functionality.

7.1 Client Sending a Message

In this example an application will be built that sends a Client message to a WCTP server and receives a success or failure response. This application is non-graphical and must be run from DOS or a UNIX shell prompt.

```
import java.net.*;
import java.io.*;

import com.arch.wctp.*;

/**
 * SendTestMessage
 * This class is used to send a Submit Client Message to a WCTP gateway.
 * Creation Date: 7-31-2001
 */
class SendTestMessage {
        private HttpURLConnection wctpConnection = null;
        private WctpClientOperations clientOps = null;
        private WctpClientReceive clientRx = null;

        private String trackNo = null;

/**
 * This method sends a Client message to a WCTP gateway.
 */
private void sendMessage(String gateway, String deviceID, String message,
                                String senderID) {
        // Instantiate the ClientOperations class
        clientOps = new WctpClientOperations();

        // Build the operation
        String body = clientOps.submitMessage(clientOps.getTimestamp(),
                                deviceID, message, senderID, true);

        // Set up the URL
        URL archWireless = null;
        try {
                archWireless = new URL(gateway);
        }
        catch (java.net.MalformedURLException mue) {
```

```
                System.out.println("Unable to connect to specified URL");
        }

        // Create a new HttpURLConnection object to use for communication
        // Note that this connection is reused in the getResponse() method
        wctpConnection = clientOps.sendWctpPacket(archWireless, body);
}

/**
 * This method is used to get a response from a WCTP server.
 */
private void getResponse() {
        // Instantiate the receive class
        clientRx = new WctpClientReceive();

        // Get back the success or failure response from the server
        String response = clientRx.readResponse(wctpConnection);

        // Parse the response for the success or failure code
        int responseCode = clientRx.getResponseCode(response);
        String codeText = clientRx.findCode(responseCode);

        // Save the tracking number for future use
        trackNo = clientRx.getTrackingCode(response);

        // Output the three variables
        System.out.println("The response code is: " + String.valueOf(responseCode));
        System.out.println("The response text is: " + codeText);
        System.out.println("The tracking code is: " + trackNo);
}

/**
 * This is the main method of the application.
 */
public static void main(java.lang.String[] args) {
        SendTestMessage stm = new SendTestMessage();

        // Prompt the user for the gateway information
        String deviceID = null;
        String mssg             = null;
        String gateway  = null;
        String senderID = null;

        try {
                BufferedReader in = new BufferedReader(
                        new InputStreamReader(System.in));
```

```
                // Get the gateway URL
                System.out.println("\nEnter the gateway URL:");
                gateway = in.readLine();

                // Get the Device ID
                System.out.println("\nEnter the Device ID:");
                deviceID =      in.readLine();

                // Get the Sender ID
                System.out.println("\nEnter the Sender ID:");
                senderID =      in.readLine();

                // Get the message text
                System.out.println("\nEnter the message text:");
                mssg = in.readLine();

                // Send the message
                stm.sendMessage(gateway, deviceID, mssg, senderID);

                // Get the response
                stm.getResponse();

                // Prompt the user to see if he/she would like to check for a receipt
                // for the previous message
                //System.out.println(
                //        "\n\nWould you like to check for a receipt? (Y or N):");
                // String receipt = in.readLine();
                // if(receipt.equals("Y")) {
                //        stm.lookForReceipt(deviceID, senderID);
                // }
        }
        catch(IOException e) {
                System.out.println("Error reading device ID");
        }
}
}
```

7.2 Client Getting a Delivery Confirmation

This section builds on section 9.1 to add a ClientQuery to check for a receipt. You will probably want to wait until your device receives a message for your initial test of the code.

Near the end of the main() method in the class SendTestMessage, change the following lines of code:

```
// Prompt the user to see if he/she would like to check for a receipt
// for the previous message
//System.out.println(
//        "\n\nWould you like to check for a receipt? (Y or N):");
// String receipt = in.readLine();
// if(receipt.equals("Y")) {
//        stm.lookForReceipt(deviceID, senderID);
// }
```

To this:

```
// Prompt the user to see if he/she would like to check for a receipt
// for the previous message
System.out.println(
        "\n\nWould you like to check for a receipt? (Y or N):");
String receipt = in.readLine();
if(receipt.equals("Y")) {
        stm.lookForReceipt(deviceID, senderID);
}
```

Next, add this method into class SendTestMessage:

```
/**
 * This method checks to see if a message has been delivered.
 */
private void lookForReceipt(String deviceID, String senderID) {
        // Before the inital check for a receipt, pause to give the message
        // a chance to get delivered


        // Instantiate new instances of the WCTP Factory classes
        clientRx = new WctpClientReceive();
        clientOps = new WctpClientOperations();

        // First, build a new Client Query
        // Note that trackNo came from the getResponse method
        String body = clientOps.clientQuery(deviceID, trackNo, senderID);

        // Set up the URL
        URL archWireless = null;
        try {
                archWireless = new URL("http://wctp.arch.com/wctp");
        }
        catch (java.net.MalformedURLException mue) {
                System.out.println("Unable to connect to specified URL");
```

```
        }

        // Create a new HttpURLConnection object to use for communication
        // Note that this connection is reused later to get the response from the server
        wctpConnection = clientOps.sendWctpPacket(archWireless, body);

        // Now we will need to get a response from the server
        String response = clientRx.readResponse(wctpConnection);

        // This will parse the response for a type of DELIVERED
        boolean delivered = clientRx.checkIfDelivered(response);

        // Output the delivery status
        if (delivered == true) {
                System.out.println("\n\nMessage Delivered");
        }
        else {
                System.out.println("\n\nMessage Was Not Delivered");
        }
}
```

7.3 Enterprise Client Sending a MCR Message

This application sends a MCR message to a device using an Enterprise message. Because the message is an enterprise request, a device ID can be set in the sender ID field.

```
import java.net.HttpURLConnection;
import java.net.URL;
import java.io.*;

import com.arch.wctp.*;

/**
 * This class sends an enterprise MCR message.
 * Creation date: (8/2/2001 9:49:44 AM)
 */
class SendMCR {
        private HttpURLConnection wctpConnection = null;
        private WctpEnterpriseOperations entOps = null;
        private WctpClientReceive clientRx = null;

        private String trackNo = null;
/**
 * This method is used to get a response from a WCTP server.
 */
private void getResponse() {
        // Instantiate the receive class
```

```
        clientRx = new WctpClientReceive();

        // Get back the success or failure response from the server
        String response = clientRx.readResponse(wctpConnection);

        // Parse the response for the success or failure code
        int responseCode = clientRx.getResponseCode(response);
        String codeText = clientRx.findCode(responseCode);

        // Save the tracking number for future use
        trackNo = clientRx.getTrackingCode(response);

        // Output the three variables
        System.out.println("The response code is: " + String.valueOf(responseCode));
        System.out.println("The response text is: " + codeText);
        System.out.println("The tracking code is: " + trackNo);
}
/**
 * This is the main method of the application.
 */
public static void main(java.lang.String[] args) {
        SendMCR smcr = new SendMCR();

        // These fields are needed for the WCTP packet
        String deviceID = null;
        String mssg             = null;
        String gateway  = null;
        String senderID = null;
        String [] choices = new String[3];
        String sendResponsesToID = null;

        try {
                BufferedReader in = new BufferedReader(
                        new InputStreamReader(System.in));

                // Get the gateway URL
                System.out.println("\nEnter the gateway URL:");
                gateway = in.readLine();

                // Get the Device ID
                System.out.println("\nEnter the Device ID:");
                deviceID =      in.readLine();

                // Get the Sender ID
                // This will indicate where the MCR response should go
                System.out.println("\nEnter the Sender ID:");
```

```
                senderID =      in.readLine();

                // Get the Message Text
                System.out.println("\nEnter the message text:");
                mssg = in.readLine();

                // Get MCR choice #1
                System.out.println("\nEnter the first MCR:");
                choices[0] = in.readLine();

                // Get MCR choice #2
                System.out.println("\nEnter the second MCR:");
                choices[1] = in.readLine();

                // Get MCR choice #3
                System.out.println("\nEnter the third MCR:");
                choices[2] = in.readLine();

                // If a notification is desired, set the variable to an application URL
                System.out.println(
                        "\nWould you like a notification sent to an application? (Y or N)");
                String notify = in.readLine();
                if(notify.equals("Y")) {
                        System.out.println(
                                "\nPlease indicate an application to send the response to");
                        sendResponsesToID = in.readLine();
                }

                // Send the message
                smcr.sendMessage(gateway, deviceID, mssg, senderID, choices,
                        sendResponsesToID);

                // Get the response
                smcr.getResponse();
        }
        catch(IOException e) {
                System.out.println("Error reading device ID");
        }
}
/**
 * This method sends a Enterprise MCR message to a WCTP gateway.
 */
private void sendMessage(String gateway, String deviceID, String message, String
senderID,
                                                String [] choices, String
sendResponsesToID) {
```

```
        // Instantiate the EnterpriseOperations class
        entOps = new WctpEnterpriseOperations();

        // Build the operation
        String body = entOps.submitMcrRequest(deviceID, "Arch87965Q", senderID,
                message, choices, sendResponsesToID);
        System.out.println(body);

        // Set up the URL
        URL archWireless = null;
        try {
                archWireless = new URL(gateway);
        }
        catch (java.net.MalformedURLException mue) {
                System.out.println("Unable to connect to specified URL");
        }

        // Create a new HttpURLConnection object to use for communication
        // Note that this connection is reused in the getResponse() method
        wctpConnection = entOps.sendWctpPacket(archWireless, body);
}
}
```

7.4 Device Originated Message to an Application

In this example, a device application sends the string “Hello WCTP Application” to an application at http://localhost:80/ExampleDatabaseApp. *Please note: the actual construction of device applications is well beyond the scope of this manual.* Our servlet will capture the WCTP payload and write the string to an HTML page that we can view on the web. The servlet also sends a message to a device to identify to a remote user that a Status Update has been received.

```
package net.wctp.proxy;

import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;

import com.arch.wctp.*;

import java.net.*;

/**
 * This class is a servlet that gets a POST from a WCTP server.
 * The input is checked to see if a message is delivered.
 * Creation date: (8/1/2001 3:04:22 PM)
```

```
 */
public class WctpPageBuilder extends HttpServlet {
	private WctpEnterpriseOperations entOps = null;
/**
 * This method handles the HTTP post from the WCTP server
 * @param req the HTTP request
 * @param res the HTTP response
 * @exception ServletException
 * @exception java.io.IOException
 */
public void doPost(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException {
	res.setContentType("text/xml");
	PrintWriter out = res.getWriter();

	entOps = new WctpEnterpriseOperations();

	// Parse the XML packet
	StringBuffer inputXML = null;
	try {
		BufferedReader in = new BufferedReader(new InputStreamReader
			((InputStream)req.getInputStream()));
		inputXML = new StringBuffer();
		String input = null;
		while ( (input = in.readLine()) != null) {
			inputXML.append(input);
		}
		in.close();

		// return a success response
		out.println(entOps.failure("200", "Success", "Status Info Recieved"));

		// Send a confirmation page back to a device
		sendMessage();
	}
	catch(Exception e) {
		// Give a generic error
		out.println(entOps.failure("301", "Parse Error",
			"There was an error while parsing the XML"));
	}


	// Find the payload.  This example is designed for a simple example.
	// A better solution here would be to parse the input using Xerces
	// or another XML parser.
	String payload = getPayload(out, inputXML.toString());
```

```
		// Write the output to a file
		outputHTML(payload);
}

/**
 * This method strips out the payload from a body of text.
 * This method is "quick n dirty" to reduce the size of the
 * source code.  A better solution would be to use an XML parser
 * to parse the data.
 * @param inputXML the XML received by the gateway
 * @return java.lang.String
 */
private String getPayload(PrintWriter out, String inputXML) {
		String payload = null;
		try {
			int i = inputXML.indexOf("<wctp-Alphanumeric>");
			int j = inputXML.indexOf("</wctp-Alphanumeric>");
			if (i != -1) {
				payload = inputXML.substring(i+19, j);
			}
		}
		catch (StringIndexOutOfBoundsException e) {
				System.out.println("Nothing Received?");
				// Give a generic error
				out.println(entOps.failure("301", "Parse Error",
					"There was an error while parsing the XML"));
		}
		finally {
			return payload;
		}
}

/**
 * This method writes the output to a file used by a webserver.
 * @param out the PrintWriter object
 * @param payload The payload received from
 */
private void outputHTML(String payload) {
		try {
			File outputFile = new File("/var/apache/htdocs/WctpOutput.html");
			FileWriter out = new FileWriter(outputFile);

			out.write("<html><body>" + payload + "</body></html>");

			out.close();
```

```
        }
        catch (IOException e) {
                System.out.println("Error writing file: " + e );
        }
}


/**
 * This method sends a message to a pager.
 */
private void sendMessage() {
        // Instantiate the ClientOperations class
        WctpEnterpriseOperations entOps = new WctpEnterpriseOperations();

        // Set up the URL
        URL archWireless = null;
        try {
                archWireless = new URL("http://wctp.arch.com/wctp");
        }
        catch (java.net.MalformedURLException mue) {
                System.out.println("Unable to connect to specified URL");
        }

        // Build the operation
        String body = entOps.submitRequest("INSERT_DEVICEID",
                "WctpPageBuilder received a post from WCTP",
                "INSERT_EMAIL", null, "TestMessageID-1234");

        // Create a new HttpURLConnection object to use for communication
        // Note that this connection is reused in the getResponse() method
        HttpURLConnection wctpConnection = entOps.sendWctpPacket(
                archWireless, body);

        // Read the response from the server (This also closes the response)
        // For simplicity, the response will not be parsed.  We just assume that the
        // message was sent.
        String response = entOps.readResponse(wctpConnection);
}
}
```

8 Example Enterprise Application Framework

The following is an example of a proxy model that is currently being used in some enterprises to route WCTP traffic to the Arch gateway. A limited functionality reference implementation is provided on the CD included with this manual.

8.1 Proxy Model

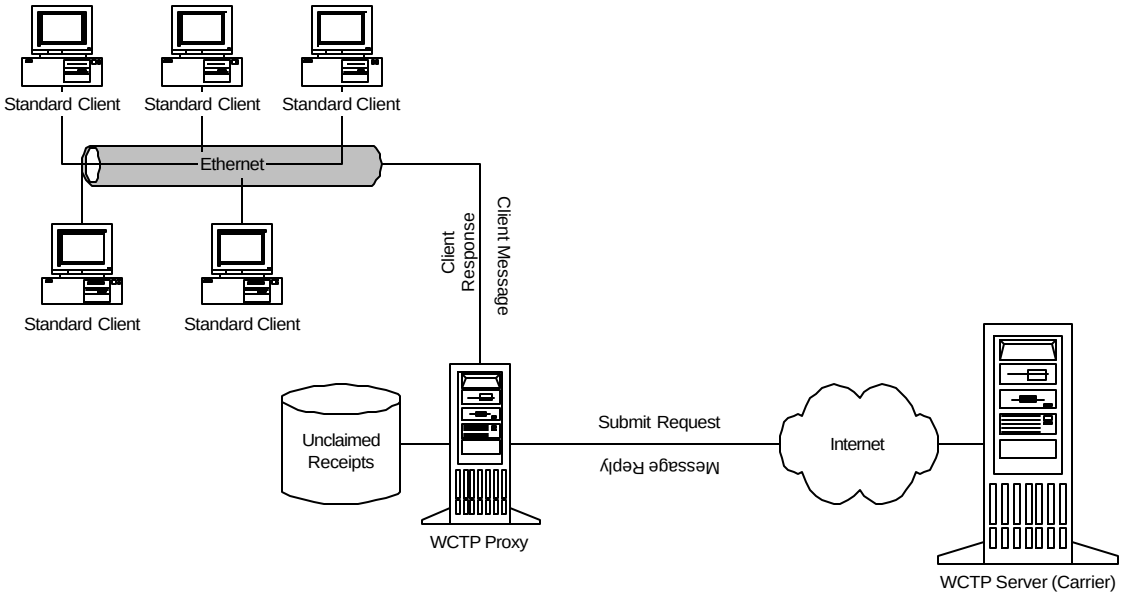


Figure 8-1 Client Message to Enterprise Request

The proxy receives a message from a transient client and creates a wctp-SubmitRequest packet that is sent to the WCTP Server. The senderID and messageID fields are transposed from the wctp-ClientMessage packet to the wctp-SubmitRequest packet. The WCTP Server responds with a wctp-MessageReply packet. The proxy receives the reply and generates the appropriate message to the transient client. Figure 10-1 shows transient client on a LAN generating wctp-ClientMessage packets to a proxy.

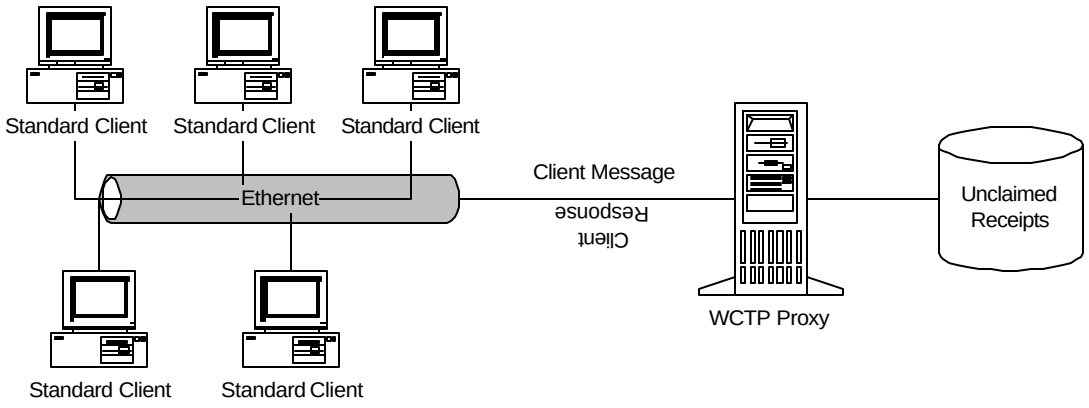


Figure 8-2 Client Queries

A transient client queries the proxy using the wctp-ClientQuery operation. The proxy checks its database and responds with a wctp-ClientQueryResponse packet. The proxy stores all client query requests as undelivered receipts until a receipt has been provided or until it expires due to time limits. Figure 10-2 shows transient client on a LAN generating wctp-ClientQuery packets to a proxy.

8.2 Push Based Model

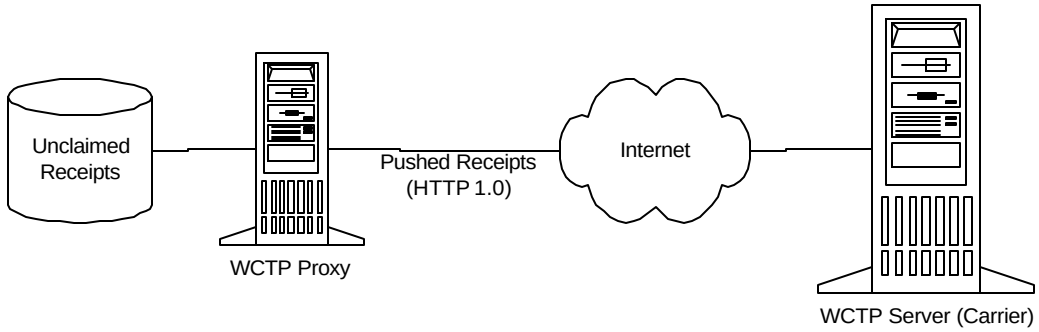


Figure 8-3 Proxy Receiving Pushed Responses

The preferred model for a proxy to obtain receipts from a WCTP Server is through a push mechanism. The proxy simply listens on a port for receipts being posted to that port using the HTTP protocol. The proxy updates records in the database with delivery confirmations at the time when receipts are received.

8.3 Central Database for the Proxy

Database wctp_receipts

receipts_mst

PK			FK		
char(50)	*char(50)*	*char(256)*	*int*	*char(256)*	*Char(19)*
track_no	message_id	sender_id	notification	recipient_id	timestamp
1234.23xz	1234.23xzArch	randy@arch.com	1	1234567	2000-03-1T19:45:00
1234.239z	1234.239zArch	joe@arch.com	2	7654321	2000-03-1T19:45:10
1234.982z	1234.982zArch	sue@arch.com	3	5545545	2000-03-1T19:46:22

notification_mst

PK	
int	*char(10)*
type	descr
1	Queued
2	Delivered
3	Read
4	Unknown

Table 1: Database Definition

Table 1 shows an example of a database structure used in the proxy model.

8.3.1 Inserting a Receipt

At the time that the proxy receives a response from the WCTP server the proxy detects if the transient client has requested a receipt and, if so, inserts a new record into the database. A unique key is generated that is duplicated and appended with the enterprise's name. The initial notification type is set as "Unknown."

If the response was a failure, an entry is not made in the database. After that point, the client will receive a failure or success response from the proxy based on the response from the server.

8.3.2 Querying a Receipt

A receipt is queried in the database when a transient client performs a wctp-ClientQuery.

8.3.3 Updating a Receipt

A receipt is updated in the database when new status updates are received from the WCTP server. This happens when a wctp-StatusInfo operation is pushed to the proxy servlet.

8.3.4 Purging the Database

The database can be purged at the discretion of the proxy administrator. The carrier, however, is likely to have a predefined set of rules on how long they will continue to try to deliver a message. For example, the carrier may give up on a delivery confirmation after 72 hours. If the proxy does not purge the database periodically, the database could experience problems due to an unexpectedly fast growing table.

8.4 Using a Proxy in the Enterprise

8.4.1 Distributed Applications

The proxy model offers a way to distribute client applications in the enterprise to allow for easy updates and increased control over wireless traffic. Because the applications are communicating over XML, the clients are entirely platform independent. The client may be a Visual Basic application, web-based application, or any other language that has the ability to communicate using sockets. This model allows ultimate flexibility in controlling the flow of wireless traffic and monitoring the content of the traffic.

8.4.2 Access Control

One possibility for a proxy is to restrict access to a group of device ID's or to limit the amount of characters that may be sent in a single page. A proxy may also be set up to limit the amount of pages that a particular sender may send in a day. Let's take an example scenario to look at how these ideas may be used in a real-world application. Widgets Inc. has 50 employees that carry two-way wireless devices. 10 of those employees are management and need to have unlimited use of the device. So, the proxy administrator sets up a group that is allowed to send 100 messages per day with 500 characters per message to any valid device ID for the management managers.

A second group is set up for five employees who dispatch information to field technicians. They also have a need to send one-hundred messages per day, but they only need the ability to send to twenty field technicians and five managers. The proxy administrator constructs a second group for the dispatchers. A third group is set up for 20 field technicians. This group is restricted to fifty messages per day and they can only be sent to dispatchers. Of course, this is only a simple example. The possibilities for similar applications are endless.

8.4.3 Logging

Another possibility with a proxy is to add the ability to log the content of messages. The incoming XML can be written to a file to be later reviewed. The file may be written in a format that lends itself to reporting software.

9 Where to Find the Protocol

The protocol can be found at http://www.wctp.org/download.html. Read and agree to the terms and conditions. Two main documents are available. The main protocol specification defines the protocol. A use case companion is also at this site that provides XML use case examples of WCTP.

10 Commercial Software

The following companies are known to produce software that uses WCTP.

Company	URL
InfoRad, Inc.	http://www.inforad.com
OnWOW, Inc.	http://www.2bpaged.com/
SilverLake Communications	http://www.silverlake2000.com/productinfo/airsourceweb.htm
Telamon	http://www.telamon.com/

11 Available API's

The following is a list of known API's for WCTP:

API	Language	URL
Arch Wireless WCTP Factory	Java, C++	http://content.arch.com/developer
Using WCTP to send XML Forms over HTTP	Python	http://sourceforge.net/projects/wctpxml-python
WctpXML (WCTP Toolkit)	C++	http://freshmeat.net/projects/wctpxml/
g-page	C	http://sourceforge.net/projects/g-page/

12 What's included on the CD?

The following Arch Wireless APIs:

- ?? WCTP API for Java
- ?? WCTP API and for C++ (RedHat Linux 6.2 Binaries)
- ?? WCTP ActiveX Control

The following sample applications and documentation:

- ?? Arch Wireless Sample Enterprise Application
- ?? Arch Wireless Proxy Servlet for the Servlet 2.1 API
- ?? Timeport P935 Binary Test Application w/ Source Code
- ?? All source code presented in this manual.
- ?? This manual in PDF format.