Winlink Data Flow and Data Packaging

Winlink Development Team, 2017

Client Software and Message Entry

Getting Started

The Winlink Express client program is used to compose, read, and manage messages. Winlink Express is a free program, openly available for download by any licensed ham from www.winlink.org. There is no cost for the software or for using the Winlink system to transfer messages.

After installing Winlink Express, the user is guided to a setup screen where they enter their callsign, system password, grid square, and other configuration information. After submitting this information, a Winlink account is established for the user. The requirement for an account password prevents a rogue user from hijacking an authorized account. Callsigns are validated by the Winlink system to make sure they are legitimate and legal.

Message Entry

Winlink Express is similar to Microsoft Outlook and other e-mail programs. The user clicks an icon to begin composing a message. Then the text is entered and file(s) are attached to the message if desired. Once the message is completed, it is posted to the “Outbox” where it waits to be sent. The “Outbox,” “Inbox” and other message storage folders are databases stored on the user’s computer. A message placed in the Outbox is held until the user opens a sending session. After sending messages, the message status is changed to “sent,” but the message remains in the database for later reference.

Message Sending/Receiving Sessions

When the user is ready to send pending messages and check for incoming messages, they open one of the of available session types. Winlink Express provides multiple types of sessions including telnet (non-RF), Pactor, Packet, Winmor, ARDOP, Vara, and Iridium. Once a session has been opened, the user clicks “Start” to initiate a connection to a server.

HF sessions such as Pactor and Winmor have built-in busy-channel detectors that pop up a warning box if they detect a signal on the frequency about to be used. While this warning is helpful, the user is expected to listen to audio on a frequency to determine if a channel is busy before starting to call.

After Start is clicked, Winlink Express issues hailing calls in the appropriate transmission mode. If the selected station hears the call and is free, it will respond, and handshaking will take place to verify the user’s callsign and password. After this validation, any pending messages in the Outbox are sent. Then the session receives any pending incoming messages for the callsign. Once message sending and receiving are completed, the session closes the connection to the radio server and ceases transmitting.

Message Transfer Protocol and Packaging Layers

Establishing a Connection

After a call has been accepted, Winlink Express sends the user's callsign and a SID line containing information about the name of the client program (Winlink Express) and its capabilities.

The station responds by sending a ";PQ: *nnnnnnnn*" line back to the client where "*nnnnnnnn*" is a random string of digits. This is the *account* (*password) challenge*. Winlink Express hashes the challenge number with the user's password and sends back to the server a ";PR: *nnnnnnnn*" response where "*nnnnnnnn*" is the computed hash code of the challenge string and the password. This allows password validation without requiring any encrypted data being sent.

If password validation is successful, the station will send a greeting string, and the client/server will begin using the B2F Protocol to manage message traffic.

Open B2F -- Winlink Message Structure and B2 Forwarding Protocol

The open-source "B2F" protocol is used to coordinate message sending and receiving. This protocol is documented in detail at https://winlink.org/B2F.

Message Packaging Layers

Similar to TCP/IP, Winlink uses "layers" of protocols to wrap messages for transmission. Some layers are in the software, and some are handled within the specific transmission protocol (e.g., Pactor, Winmor, Packet). There are several reasons for these packaging protocols:

1. Add error checking information (CRC checksums) to detect any transmission errors.
2. Coordinate retransmissions when necessary to replace damaged or missing packets.
3. Break the transmission into manageable chunks for practical RF transmission.
4. Allow an interrupted download to be resumed at the point where the interruption occurred without requiring retransmitting data successfully received.
5. Perform data compression to reduce the amount of data that has to be transmitted allowing more efficient use of RF transmission time.

This document will discuss only layers handled at the software level.

MIME e-Mail Message Protocol

Each message transferred by Winlink is formatted in the Internet-standard MIME (Multipurpose Internet Mail Extensions) format. This is documented in RFC 2045 -- https://tools.ietf.org/html/rfc2045. MIME includes specifications for the sender, recipient(s), message text, and file attachments, etc.

Assembling a Message for Transmission

The first step in preparing a message for transmission is to assemble it with a header, the message text, and any attachments. The assembly has these components:

- MID: (message-ID) – The message-ID is a random string assigned by Winlink Express when the message is composed. It uniquely identifies the message during its lifetime in the Winlink system.
- Date: *date* – The date when the message was posted for sending. The date is formatted as *yyyy*/*mm*/*dd hh*:*mm*
- Type: Private – Indicating it's a personal message.
- From: *sender* – This is the callsign of the person sending the message.
- To: *address* – This is the callsign or e-mail address to which the message is being sent. If there are multiple recipients, there is a separate To: line for each one.
- Cc: *address* – If carbon copies were requested, this specifies the callsign or e-mail address where a copy is to be sent.
- Subject: *subject* – The subject of the message
- Mbo: *source* – The callsign of the person sending the message.
- X-P2P: True – Optional entry used only for peer-to-peer messages.
- Body: *length* – Specifies the length in bytes of the body of the message.
- File: *length file-name* – If files are attached to the message, there is one "File:" entry for each attachment. It specifies the length of the file (bytes) and the name of the attached file.
- (blank line) – Used to indicate the end of the header.
- (body of message) – This is the actual text comprising the body of the message.
- If there are file attachments, each binary image is added as follows:
    - A blank line (carriage-return, line-feed)
    - The binary image of the attachment. The size was specified with the "File:" line.
- If file attachments were added, a final blank line (CR, LF) is added after the last one.

Compressing the Assembled Message.

Once the message components have been assembled as described above, the assembly is compressed to reduce the amount of data that has to be transmitted. Winlink uses the same compression program used for FBB B1 compression and available as B2Compress.exe in the FBB bulletin board package. Compatible derivative forms of the compression algorithm (LZH or LZHUF) are freely available online, and its open development history can be explored by examining documentation within code sources. A formal description of the LZH format is at http://www.onicos.com/staff/iz/formats/lzh.html. The actual source code used in Winlink programs is freely available and publicly archived at https://github.com/ARSFI/Winlink-Compression. The actual compressed image has a 2-byte CRC checksum prepended on the front.

Breaking Message Data into B2 Formatted Data Blocks

During message transmission, an assembled and compressed message is broken into B2 formatted data blocks. The *first* B2 block begins with this header:

- SOH – byte with hex value 01
- *length* – Single byte containing length of the message subject + number of bytes to store the *offset* value (as an ASCII value) + 2
- *subject* – The subject of the message
- NUL – byte with hex value 00
- *offset* – If the message is starting at an interrupted point, this is the byte offset of the starting point. If the message is starting at the beginning, the offset is 0 (zero). The offset value is stored as an ASCII string.
- NUL – byte with hex value 00
- If the starting offset is not zero, this information field is added (omitted if offset is 0):
    - STX – byte with hex value 02
    - 6 – byte with hex value 06
    - *lead-bytes* – First 6 bytes of the compressed image.

After the B2 header information, the compressed message image is formatted into multiple blocks with each block having the following contents:

- STX – byte with hex value 02
- *block-length* – byte with the value 250 for a full block or a smaller number for the final, short block.
- *data* – As many data bytes from the compressed message image as specified by *block-length*. Full blocks have 250 data bytes, the final block may have fewer.

An end block is added after the required number of data blocks. The end block has this format:

- EOT – byte with hex value 04
- *checksum* – Simple checksum of the compressed data bytes in the block converted to a byte value using this VB.NET code:
  CByte(((intCheckSum And &HFF) * -1) And &HFF)