SAAMFRAM Communication Protocol

Inventor and Author: Lawrence Byng. WH6GGO

Publication Date: September 29th, 2022.

Last Revision Date: October 21st 2022.

Protocol Version 1.0

Table of Contents

1. FCC Rules § 97.309(a)(4) Technical Descriptions

FCC rules § 97.309(a)(4) reads as follows:

(4) An amateur station transmitting a RTTY or data emission using a digital code specified in this paragraph may use any technique whose technical characteristics have been documented publicly, such as CLOVER, G-TOR, or PacTOR, for the purpose of facilitating communications.

Documentation should be adequate to:

(a) recognize the technique or protocol when observed on the air,

(b) determine call signs of stations in communication and read the content of the transmissions.

2. Document Summary

This document details the SAAMFRAM v1.0 protocol and technique. Details include: how to recognize a SAAMFRAM communication, how to determine the callsigns in communication and how to read the content. For reference, this document is also posted on the internet at www.github.com. Please check the internet location for the most up-to-date information on the SAAMFRAM protocol. Also included with the document is an open source cross platform reference application called SAAM-MAIL. This application implements most of the features of the SAAMFRAM protocol and is able to send and receive ICS forms over JS8 or Fldigi to/from a group of stations using the SAAMFRAM protocol.

3. SAAMFRAM Protocol

I. Design Goals

SAAMFRAM is designed for use on ham radio bands in particular the HF band where propagation is often less than ideal. The primary design goals for this communication protocol and technique are:

1) Group Communication of Forms: Produce a data efficient protocol for sending forms to multiple stations simultaneously in a group setting with efficient round robin verification and re-transmit process.

2) Notifications: Provide an efficient mechanism for group notification of any messages waiting to be sent out

3) Flexibility: Incorporate multiple message delivery techniques including push, pull, store and forward, relay, active session, passive mode.

4) Content Only: Separate out the pre-existing form information such as form layout and text field information from the form content thus significantly reducing the amount of data that needs to be sent with any ICS form message transfer. Content only delivery of forms reduces message length significantly and increases performance, reliability and resilience.

5) Data Compression: Run Length Encoding is used for content data and Dictionary Compression for form templates. These techniques significantly reduce the amount of data that needs to be transmitted when sending forms. This provides an additional reduction to the average message length and further increases performance, reliability and resilience.

6) Increased Performance and Resilience: Multiple techniques have been incorporated. When used in combination, these provide a highly accurate and efficient RF data transfer mode that can be used with a variety of digital modes and Ham Radio bands including the lower HF bands (160/80/60/40/30m) where reliable communication can be problematic.

II. Overview

SAAMFRAM is an abbreviation of Sequental Arq Acknowledged Multiple FRAgment Message. It is designed specifically for use with digital amateur radio modes such as JS8 with the JS8Call application, as well as many other digital modes such as as DominoEX and QPSK as found in the fldigi application.

To Acheieve this it is necessary to deliver two protocols that are very similar but different in some key aspects particularly in regard to message fragmentation. This is due to the difference in the underlying digital mode modulation layers. The two protocols are referred to as SAAMFRAM-JS8 for use with JS8 mode and JS8Call and SAAMFRAM-General for use with the many other digital modes and the fldigi application. SAMMFRAM-JS8 is optimized for JS8.

The SAMMFRAM protocol spans multiple function layers. This can best be explained by cross referencing the OSI model in relation to the different aspects of the SAAMFRAM protocol. SAAMFRAM layers are analagous to OSI model layers 4 thru 7 (Transport layer, Session Layer, Presentation Layer and Application Layer). When sending a transmission, the process starts at the application layer with interactions by the station operator, this proceeds to the presentation layer where the form and form data entered by the operator is compressed and reformatted. This then proceeds to the session and transport layers where the final data is sent for transmission; the session layer manages the addressing of the various stations in the group and the transport layer segments and checksums the information so that it can be reliably transferred to the various stations in the group. At the other end the receiving station follows this same process but in reverse; the accurate receipt of the data from the transport layer is managed by the transport and session layers. This then proceeds to the presentation and application layers where the data is reconstituted, un-compressed and presented to the station operator. While this document describes the details of the SAAMFRAM protocol, on occasion for purposes of clarity it is necessary to reference the corresponding SAAM-MAIL application OSI Layer functionality.

The protocol is specifically designed for sending form messages to a group of stations using an Automatic Repeat reQuest (ARQ) model so that each individual station can acknowledge accurate receipt of the data or request a re-transmit. The benefits of this approach are that when delivering messages to multiple stations, each transmission is received, decoded and saved by all stations in the group. This potentially reduces the total number of back and forth re-transmission attempts for any individual station.

Each transmission is divided up into a number of fragments that are validated by the receiving station for accuracy. CRC checksums are placed on the end of each fragment (SAAMFRAM-General format) as well as on the end of the complete message (both

SAAMFRAM - General and SAAMFRAM JS8). SAAMFRAM JS8 uses the underlying frame checks of JS8 itself rather than placing a checksum on each fragment. If JS8/JS8Call detects a dropped frame then the data impacted by the dropped frame is discarded and subsequently requested for re-transmission.

If the message is addressed to multiple recipients, following the initial transmit to the group, each recipient station is addressed individually in a round-robin manner. This validation phase of the message transfer is so that re-transmits can occurr as required. Once each station has been addressed the session ends and the sending station sends out a end of session (EOS) message to close out the session.

The protocol uses the ASCII character set and makes use of multpiple compression techniques including dictionary compression for form data, Run Length encoding for content data and NACK sequence data. Additionally, the protocol separates the pre-defined form data from the content data so that a message can be delivered by sending only the content portion of the message. When combined with the earlier described ability to receive passively as well as during the active session based transfer, these approaches significantly reduce the average message length and number of retransmits required to deliver messages to a group. The result is improved performance, efficiency and increased resilience to adverse band conditions.

When used in conjunction with JS8/JS8Call it is also possible to receive form transmissions passively from multiple stations simultaneously. These capabilites make it possible to send form messages Peer to Peer, Peer to Group, Group to Peer and Group to Group. These capabilities provide the basis for a new set of tools for ham radio operators to complement the existing Peer to Peer and Peer to Gateway approaches.

III. SAAM-MAIL Application Reference Platform

The initial release of SAAMFRAM is accompanied with an open source, cross platform application written in python called SAAM-MAIL. The application implements many of the aspects of the SAAMFRAM protocol and can be used as a tool for transferring forms and form data including ICS forms. The application is currently in beta release.

4. Application Layer (OSI Model Layer 7)

At the application layer there are multiple possible interactions by the station operator with the protocol. These interactions include commands, specifying the various transmission parameters as well as selecting which optional components are to be included in the messages transmitted. The various application layer commands are as follows:

I. Application Layer Commands:

Command	Description	Format
SAAM	SAAM command announces to other stations, your presence on frequency as well as having SAAM capability	<From Call Sign>: <Group Name> SAAM <optional CK-IN or QRT designator> <From Call Sign>
SAAM?	Command sent to a single station to determine if that station is present	<From Call Sign>: <Call Sign> SAAM? <From Call Sign>
SAAM?	Command sent to the group to ask stations to come back with a SAAM response if they are present and can hear the signal	<From Call Sign>: <Group Name> SAAM? <From Call Sign>
REQM	Command is used to request a critical message from the specified sending station. This is used when only the stub (message id and receive list) have been received by the receiving station.	<From Call Sign>: <Call Sign> REQM <MSGID> <From Call Sign>
REQF	Command is used to request critical message fragments from the specified station. The message is similar to REQM but also specifies the fragmentation size so that any resend of fragments will use this same size to complement any existing fragments that may have already been received.	<From Call Sign>: <Call Sign> REQF <MSGID> <FRAGMENT SIZE> <FRAGMENTS> <From Call Sign>
REQC	Command is used to request an alternate set of CRC checksums for a given message id and fragment size including the end of message (EOM) checksum.	<From Call Sign>: <Call Sign> REQC <MSGID> <FRAGMENT SIZE> <From Call Sign>
CSUM	Command is used to deliver the alternate sequence of checksums	<From Call Sign>: <Call Sign> CSUM <MSGID> <FRAGMENT SIZE> <CHECKSUMS>
QRYM	Command is used to ask if a station or group has a full verified copy of the message identified by message id and is able to forward this message.	<From Call Sign>: <Call Sign> QRYM <MSGID> <From Call Sign> <From Call Sign>: <Group Name> QRYM <MSGID> <From Call Sign>
QRYF	Query if the call sign or group has the specified fragments	<From Call Sign>: <Call Sign> QRYF <MSGID> <FRAGMENTS> <From Call Sign> <From Call Sign>: <Group Name> QRYF <MSGID> <FRAGMENTS> <From Call Sign>
CONF	Conf command confirms that a station has a copy of the message with a given message id ready to send. The requesting station can then issue REQM, REQF or REQC messages to that station as required.	<From Call Sign>: <Call Sign> CONF <MSGID> <From Call Sign>
RDY?	Is the station ready to receive message identified by MSGID?	<From Call Sign>: <Call Sign> RDY? <MSGID> <From Call Sign>
NR	Not ready to receive message	<From Call Sign>: <Call Sign> NR <MSGID> <From Call Sign>
RR	Ready to receive message identified by MSGID.	<From Call Sign>: <Call Sign> RR <MSGID> <From Call Sign>
CCL	Cancel already have a copy	<From Call Sign>: <Call Sign> CCL <MSGID> <From Call Sign>

i. Querying Relay Stations.

Relay stations can be used to re-transmit missing fragments as well as to relay entire messages. To achieve this, the process involves first sending out a query (QRYM or QRYF) to see which stations are able to relay the message or fragments for a message with a given message ID. Stations in posession of a full copy (for QRYM requests) or only the fragments (for QRYF requests) and who are also ready to transmit, can then reply back with a confirm (CONF). On receipt of the CONF the station requesting the message or fragments can then address the station directly to request the full message or message fragements as required using REQM or REQF.

II. Other Application Layer Interactions

Based on information received from interactions at the application layer, the station operator may have multiple stations listed as available for communication and message transfer.

Additional station operator specified interactions are as follows:

- Station operator selects which sending station is currently being listened to for the active message transfer of the critical message.
- Station operator determines the digital modulation format to be used i.e. JS8 or DominoEx as well as frequency and offset within the pass band.

The format of the messages is also under the direct control of the station operator. The formats and the different options are as follows:

- Critical Message Format: Content only format or Template + Content format
- Pre-Message: This option includes a pre-message on the front of the critical message. The pre-message is a passive mode only message with a checksum, allowing receiving stations to validate accurate receipt of the information.
- The type of pre-message to be sent. For example PEND will list stub information (message id and receive list) for messages that are waiting to be sent (outbox messages) or stored messages waiting to be forwarded (Relay box messages). If there are multiple pending messages then the specific message will be chosen at random and its message id and receive list included in the PEND pre-message. All of the different sub-options for the pre-message are detailed in the later section.
- Repeat Message – Used to repeat the full set of message fragments and to interlace these with the original message.
- Repeat Fragment – Used to repeat upto the first three fragments of a message upto three times at different points in the message. The first few fragments of the critical message contain the message id and receive station list. These can be used by other stations to later ‘pull’ the remainder of the message if the initial ‘push’ mode does not succeed.
- Fragment Size - This is the length of the individual fragments used in the critical message. Each fragment contains the specified number of characters of the message. The last fragment length may be shorter as this is the remainder.

III. Message Delivery Techniques

SAAMFRAM is specifically designed for sending forms to a group efficiently and reliably. There are two main techniques for transferring these messages:

1) Active Send Session. This involves sending the form to the group and verifying each station in turn and is synonymous to pushing the message to the group. In push mode the master station initiates message transfer to a designated group of receiving stations each of which communicates back to the master station in turn if the message was received accurately or if any parts need to be resent. A message is sent to the group along with a beginning of session (BOS) indicator at the start of the message. One of the fields on the message is a semicolon delimited callsign list of stations that the message is addressed to. The sending station will send the message to the group first and then go round robin through the list of receiving stations to check they received the message successfully or not. If required, the fragments that were not received accurately are re-transmitted. If a station does not respond then the process continues on to the next recipient on the list until all recipients have been addressed. At the conclusion of the process an end of session message EOS is sent to notify the group the session has ended.

2) A station may have received only the message id and the call list (Stub or Partial message) and is able to use this information to request that message is sent to the station. This is synonymous to pulling the message from a station that has a full copy or verified fragments of the message to send. In pull mode any station can request any message part or part(s) upto and including the full message from any other station thas has the relevant copy of the message/fragments to relay.

5. Presentation Layer (OSI Model Layer 6)

This layer handles the data compression. This uses a combination of run length encoding of the form data as well as dictionary compression of the form templates resulting in a highly efficient data transfer format. The result is reduced data transfer times and increased reliabilibility. Additionally, the forms are separated out into two components representing:

a) The blank form text and layout aspects of the ICS form and

b) The data content of the form.

In so doing, this allows for standard well defined forms such as ICS forms to have a set of templates already saved on both the sending and receiving stations thus removing the need to always send the form layout and blank form text information. For Pre-defined forms such as ICS forms, sending only the data content is possible. This results in a significant reduction to the size of the data to transfer which also translates to increased performance; reduced transmission times and increased reliability. Blank form data templates can be transmitted to the receiving station but this is not necessary for pre-defined forms. The reference application SAAM-MAIL comes with a set of pre-defined ICS forms.

I. Message Components

The various component parts of the message are easily identifiable from the ASCII characters used. These components are detailed below:

i. Message Encapsulation Brackets

The critical message is enclosed within squiggly brackets as follows {<Critical Message>}. Critical message can be either form content data or form content + form template data.

ii. DATA – Form Content

This designates the following information is content data of an ICS form. DATA follows immediately the opening squiggly bracket

Example

{DATA….}

iii. FORM – Form Template

This designates that the following information is form template information of an ICS form. FORM follows immediately the opening squiggly bracket

Example

{FORM….}

iv. Delimeter character

The tilde ~ character is used to delimit the different fields in the critical message. For example the sequential fields of form content data would be represented as follows:

~Lawrence~Puna QTH~Operator~

v. Receive List

This is a list of callsigns separated by semicilons that designate the list of station to which the message is addressed. The receive list is part of the delimited information and has a delimeter character on either side.

For example

~WH6GGO;WH6ABC;WH6DEF~

vi. Message IDs

The message Ids are unique to a single callsign to the nearest second of time within the year.

The first part of the message id is based on the callsign of the station creating the message. This is calculated by first converting the callsign into a number. This is done by multiplying the numeric value of each character in the callsign, using base 36 (0-9 and A-Z), by 36 to the power of the character's location in the callsign and adding all of these numeric values together for the full call sign. This number is then converted to hexadecimal.

The second half of the message ID is calculated by taking the numeric value of the current zulu time to the nearest second excluding the year (In reality messages will not be retained for more than 12 months and the second part of the message ID will wrap around after 12 months) and converting this numeric value to hexadecimal.

Each of the two hexadecimal parts of the message are then joined together using an underscore _ character. This makes for a very effective Message ID.

For the complete python program code for this algorithm please see appendix B.

Example of a mesasge ID is as follows:

750cdeca_3731a4b5

There is a second form of message ID that is used when replying to messages. The second form comprises two message Ids joined together using an underscore _ character. The first message ID is the ID of the original message. The second ID is the ID of the reply message. By using this approach, all messages relating to the same original message can be grouped together by the receiving station to provide a very convenient grouping and ordering of the messages when viewed.

vii. Escape Characters

There are several ASCII characters that are used by the protocol to provide sequencing of the fragments and field delimiting. In order for these characters to be used in the content of any message, it is necessary to escape these using the designated '/' forward slash escape character. In reality these characters are rarely used in content so the additional overhead of having to specify these using two or more characters instead of one is negligible.

The following escape characters are defined:

- /A This represent the '[' open square bracket character
- /B This represents the ']' close square bracket character
- /F This represents the '~' tilde character
- // This represents the '/' forward slash character
- /C This represents the '{' open squiggly bracket character
- /D This represents the '}' close squiggly bracket character
- /E This is a control character representing the end of the message in the JS8 format
- /N This is a platform neutral representation for a new line. Please note Windows uses '\r\n' characters to denote a new line, whereas under linux platform, only the '\n' character is used to denote a new line

viii. Run Length Encoding (RLE)

Run length encoding is utilzed in two areas:

1) During the sending of Content.

   In instances where there are many blank content fields on a form, these will appear in the data stream as a series of tilde characters i.e. ~~~~~~~~~~. In order to compress this data, the multiple tilde characters are replaced with a forward slash and a number followed by a tild character to indicate that the specified number of tilde characters has been replaced with the RLE code. ~~~~~~ becomes /6~

2) During the sending of NACK messages to detail the sequence of frames that need to be resent. This is described in full in the NACK session commands section of the transport layer.

Run length encoding is implemented for the tilde separator character as well as any non-escape character that is repeated 4 or more times in a row in the content. This results in a shorter message length. In the following examples, quote characters are included for clarity of explanation only and are not part of the RLE encoding. For example:

- '/42~' This is run length encoding showing a series of tilde delimeter characters. The number specifies how many characters were replaced by the RLE code. In the example 42 '~' tilde characters are represented by /42~
- '/17 ' This is run length encoding showing a series of ' ' space characters. The number specifies how many characters were replaced by the RLE code. In the example 17 ' ' space characters are represented by '/17 '

ix. Dictionary Compression

Dictionary compression is used to compress the static form template into a set of pre-defined tags. Below is an example of a complete ICS 214 form template.

{FORM~ICS~v1.0~ICS 214
Form~0B,AI~02~06~I1~42~00~06~P2~10~06~O2~D4,13,@1,D5,13~T7,13,@1,T8,13~01~02~06~N5,42~H5,42~00~06~IO,42~01~05~09,R4~N5,IO,H5~@I,0F~22,22,26~05~09,AH~D3,NB~@K,0F~22,32}

As can be seen the template is very terse. Each of the delimited fields is a tag. Tags relating to the same row of the form are separated with a comma. In general the tags beginning with a 0 or an @ character are control tags, tags beginning in a number other than 0 are entry field tags and tags beginning in a letter are text field label tags. The majority of tags fall into one of the following categories:

1) A text field with a pre-defined text string
2) A form layout field such as a data entry field, a checkbox or a radio-button group
3) A repeat line designator to repeat a line n number of times
4) Alignment indicators to align the fields on the form
5) Heading indicator for heading 1 or heading 2 information

The ICS tag file provides the basis on which all ICS form templates are built. A full tag reference for the ICS tag file is provided in appendix A.

II. Message Formats High Level

Each message has several distinct sections as follows:

<FromTo> <Non Critical Pre-Message (optional)> <Critical Msg> <EOM> <Sender Callsign>

These component parts are described as follows:

- FromTo - This is the sender callsign follwed by the receipient group name or recipient station callsign
- Pre-message - This part can take several forms as described in the section ‘posted message formats’. The section ends with a checksum for validation. This section is for passive receive only and does not participate in the re-transmit process. This section is optional.
- Critical Message - This part can take several forms as described in the section ‘critical message formats’. This section is broken into fragments and each fragmant is check-summed and can participate in an active message transfer session. This is described in the sections ‘Saamfram Fragmentation’. The last fragment ends with 4 alpha numeric checksum characters (20 bits) used as a second validation layer to validate the entire critical message.
- EOM - The section contains the text EOM which denotes the end of the message
- Sender callsign – This is present at the end of the transmission.

i. Partial Messages

Partial messages are messages where only some of the set of fragments have been received. As long as the fragments received contain at a minimum the message ID, these fragments can be retained and added to in the future either by direct requests to other stations or by listening for further transmits of the message with the same message ID during a message transfer session..

ii. Stub Messages

The following is an example of a stub message sent with the PEND pre-message. The message is only 33 characters long but contains sufficient information that a station the hears this message can determine:

1) if the message is of interest by the receive list specified
2) if the station already has a copy of the full message or not by the message id provided
3) the date and time the message was created and the creator of the message by decoding the message id
4) which station sent the message and therefore has the full message waiting to be sent / relayed.

If the station determines the message is of interest, the message can be requested from the station that sent out the PEND pre-message or from any other station that has a copy of the message by using REQM.

Example of a PEND pre-message:

PEND(750cdeca_37168699,wh6ggo,QV)

III. Pre-Message Format

During testing, it was discovered that the early parts of a message sent using some of the fldigi modes had a higher failure rate than the latter parts of the message. This was attributed to the various components needing to stabilize following the start of transmit. For this reason a non-essential variable length section was added at the start of the message to serve as a buffer before the critical message to allow the components to stabilize. Testing shows this results in a higher success rate for the critical part of the message. Secondarily, this section is used as a mechanism to post non-critical messages to the group. There can be multiple pre-message sections sent in sequence and separated with a tilde '~' character.

The pre-message is usually very short and is not fragmented. It does contain a checksum on the end for validation. The non-critical pre-message section has several formats as follows:

i. Pre-Message Format Types:

format	Description	Formats
POST	POST(...) is used to post non-critical text information to the group	POST(<Message Text>, <Checksum>) POST(<Send to Group Name>, <Message Text>, <Checksum>) POST(<Send to Station CallSign>, <Message Text>, <Checksum>)
PEND	PEND(…) is used to indicate to the group that there are other messages pending	PEND(<Message ID>, <Receive List>, <Checksum>)
QMSG	QMSG(...) is used to request details of any mssages fitting criteria for callsign or group. This can be used to store and forward messages addressed to the group	QMSG(<Station Call Sign>,<Priority>,<Days>,<Hours>,<Minutes>, <Max# Mesages>,<Checksum>) QMSG(<Group Name>,<Priority>,<Days>,<Hours>,<Minutes>, <Max# Mesages>,<Checksum>)
QINFO	QINFO(...) is used to query info of the sending station	QINFO(PHONE, <Checksum>) QINFO(EMAIL, <Checksum>) QINFO(GRID, <Checksum>) QINFO(GPS, <Checksum>)
CALLS	CALLS(…) is used to denote there are messages waiting for the following callsigns. The call sign list can include group identifiers	CALLS(<Call Sign List>,<Checksum>)
INFO	INFO(…) is used to give station info of the sending station. Can be used to provide email address, SMS info, phone# etc.	INFO(PHONE, <Phone #>, <Checksum>) INFO(EMAIL, <Email Address>, <Checksum>) INFO(GRID, <Grid Square>, <Checksum>) INFO(GPS, <GPS Coordinates>, <Checksum>) INFO(SMS, <SMS Carrier Info>, <Checksum>)

IV. Critical Message FORM - Template Format

This message format makes use of the dictionary compression technique whereby all of the text strings of the form template are represented by tags. The tags along with their associated strings have a one to one representation and are stored in a dictionary. The application provides one dictionary tag file for ICS forms named ICS. The second component of the dictionary are tags that represent the various layout and other data presentation aspects of the form such as a tag for MainHeading another for SubHeading one for repeat lines another for Start Column. These layout tags are used to specify the layout of the form template in an extremely data efficient manner.

A message can consist of fragments containing form template information and / or form content information. Generally it will only be necessary to transmit messages containing the content as the templates exist in a pre-exisiting form provided with the SAAM-MAIL application. In the event that a new form template is developed, this can be shared with other radio stations in real time using the template format.

Template + Content Format:

{FORM <TAGFILE> message}

Template Only Format:

{FORM <TAGFILE> <FORMNAME> <VERSION> message}

V. Critical Message DATA - Content Format

The form content is a sequence of values that represents data from the individual fields on the form in a sequential manner. These fields are separated with a tilde delimiter character

The content format is the same no matter if the message contains only content information or if the message also contains a template.

Content Format:

{DATA <MSGID><TO LIST> <PRIORITY> <FRAGMENT SIZE> <SUBJECT> <FORMNAME> <VERSION> message}

VI. Examples

Content only example general format...

{DATA~750cdeca_3731a4b5~WH6GGO~~10~This is a test message~ICS 214~1.3~My Test Incident~1~09-19-2022~09-20-2022~2300~2300~Lawrence~Puna QTH~Operator/45~}

Content + template example general format...

{DATA~750cdeca_3731a4b5~WH6GGO~~10~This is a test message~ICS 214~1.3~My Test Incident~1~09-19-2022~09-20-2022~2300~2300~Lawrence~Puna QTH~Operator/45~}74CI

{FORM~ICS~v1.0~ICS 214
Form~0B,AI~02~06~I1~42~00~06~P2~10~06~O2~D4,13,@1,D5,13~T7,13,@1,T8,13~01~02~06~N5,42~H5,42~00~06~IO,42~01~05~09,R4~N5,IO,H5~@I,0F~22,22,26~05~09,AH~D3,NB~@K,0F~22,32}OR1N

6. Session Layer (OSI Model Layer 5)

The session layer is involved with the following actions:

1) Starting and ending the session for the active message transfer
2) Determining the initial set of addresses of the stations that are active on the band to form an initial send list. This is the intersection set of the requested send list and the list of actual stations present on the band.
3) Managing the addressing of the messages as the active message transfer proceeds in a round robin manner through the stations on the receive list. A session can be thought of as a sequence of smaller sessions between two stations; the sending station and each of the receiving stations in turn one after the other until the transfer to all stations on the list is complete.
4) Sending out the abort message for a single station if one of the receive stations is unable to succesfully receive the message
5) Sending out an abort message to the group of stations if message transfer session is aborted
6) keeping track of multiple channels and receiving and decoding messages from these channels in a passive or sessionless message transfer process. The SAAMFRAM-JS8 with JS8Call is especially relevant in this regard. Using passive receive only process, SAAMFRAM JS8 can receive multiple simultaneous parallel message communications; a) from multiple sending stations to a single receiving station and b) from multiple sending stations to multiple receiving stations. A significant portion of the session layer functionality is relevant to the SAAMFRAM JS8 format. This format is able to keep track of multiple channels simultaneously and passively receive and assemble the messages on each of these separate channels.

I. Message Addressing

i. Call Signs

The various types of messages and their formats are summarized below using the following callsigns for illustratve purposes only: WH6ABC for station 1 and WH6DEF for station 2 and @MYGROUP for the group name. The transmission always ends with the call sign of the currently transmitting station. The following addressing formats are used during an active message transfer session:

- WH6ABC: @MYGROUP <Message> WH6ABC

  This format is used to address the group and is used by the initial message send.

- WH6ABC: WH6DEF <Message> WH6ABC

  This format is used when addressing each station individually such as during the re-transmit phase of the session

- WH6ABC ACK? WH6DEF

  This format is used to address a single station when requesting acknowledgement.

- ACK WH6DEF

  This format is used by the receiving station when replying back to the sending station with an acknowledgement.

- KCAN (<Retransmit Tag List>) WH6DEF

  This is the same format as the previous ACK. The call sign format is the same for both ACK and NACK messages.

The selection of station call signs is determined by the station operator. This selection includes choosing which station to interact with during an active communication message transfer session. Also relevant to this process is the address list of each of the receiving stations for a particular message as well as which stations are present on the band. There is obviously no point in including a station in the message transfer process based on the receive list if that station is not present on the band.

ii. Active Message Transfer

Active Message transfer requires two or more stations to participate in a message transfer session. During the active message transfer, if any fragments are not received accurately the list of erroneous fragments can be relayed back to the send station via the NACK message so that a re-transmit of the impacted fragments can occur. This part of the process is handled by the transport layer. Ideally each station should be able to send back an ACK at the end of the message transfer process to indicate message received successfully. In the event that message transfer is not possible due to band conditions or other impacts the stations can retry for a set number of retry attempts after which the transfer is abandoned / aborted for that station and the next station on the list queued for message transfer by the session layer.

iii. Passive Message Receive

Stations can receive any of the messages in a passive sessionless manner. If all of the fragments are received correctly then there is no need for any further actions to receive the message. If one or more fragments did not transfer correctly, the receiving station can at any time send a message to the group asking if any station has a full copy or specified fragments of the given message. In order to request missing fragments the receiving station must at the very least know the message id. Once a station replies back, the missing fragments can be requested from that station. This part of the process is done with the application layer interactions between the station operator and the protocol as detailed in the application layer section.

II. Session Layer Commands

Command	**Description**	**Example**
BOS	Beginning of message transfer session	WH6KLM: @HINET BOS <message> WH6KLM
EOS	End of message transfer session	WH6ABC: @MYGROUP EOS WH6ABC
ABORT	Abort message transfer for an individual station or for the entire session to the group	WH6ABC: WH6DEF ABORT WH6ABC WH6ABC: @MYGROUP ABORT WH6ABC

III. Round-Robin

After the initial message send to the group, the first station on the receive list will reply back with an ACK or NACK confirmation status. The sending station will then either re-send the missing fragments detailed in the NACK message or move on to the next station if it was an ACK message. The sending station now attempts to do the same with receiving station two. It is quite possible that receiving station two picked up additional frames during the resend process to station one if there was one. Station two is addressed directly with an ACK? message

WHABC ACK? WH6DEF

After receiving this, station two has a varying amount of time, depending on the digital mode selected, to reply back to the sending station with the ACK/NACK status. After the sending station receivies the status from the receiving station it will either re-transmit the fragments if it was a NACK message or move on to the next station and so on until all stations in the round robin send list have had their turn at message exchange. The sending station will have a complete list of which stations acknowledged receipt of all fragments and which did not. Once all stations on the list have been addressed, the sending station will send out an EOS for end of session to close out the session to the group of stations.

After the automatic send/resend process has completed there can be additional attempts at sending any missing fragments (using push or pull techniques) but these must be completed either by starting a new session or manually by the application layer commands as detailed in the application layer commands section.

IV. Example Message Transfer Session

Detailed below is an example of complete session of three stations, one sending and two receiving in an active communication session. All three stations are on frequency.

Send the initial message to the group:	WH6ABC: @MYGROUP BOS <WHDEF;WH6GHI><message> EOM WH6ABC
First stations sends back an ACK:	ACK WH6DEF
Sender asks second receive station for ack:	WH6GHI ACK? WH6ABC
Second station replies frames 3-16 missing:	KCAN (F3-16) WH6GHI
Sender resends fragment 3 thru 16:	WH6ABC: WH6GHI <specified fragments> WH6ABC
Second station replies back with ACK:	ACK WH6GHI
Sender sends out end of session message:	WH6ABC: @MYGROUP EOS WH6ABC

7. Transport Layer (OSI Model Layer 4)

The transport layer handles the task of transferring the message to the receiving station. Key to this process is segmenting the message into a sequence of fragments that are individually identifiable, can be verified and if necessary can be re-transmitted. Fragmentation together with the use of one or more checksums, provides a mechanism to ensures messages are delivered accurately and efficiently.

Messages can be delivered in an active session based communication with re-transmits and acknowledgements or in a sessionless passive communication where the receiving stations receive as many of the fragments as conditions permit. At a later stage, a station with missing fragments can request these via the various application layer interaction detailed earlier in the application layer section.

For an active message transfer process, the success or failure of the message transfer to a given receive station is relayed to the session layer for further action as the session layer handles the round robin sequencing through a list or group of receiving stations.

The passive message delivery process can be either a single send of the message or it can be from other stations, not involved in an active session, listening to the message traffic going back and forth. The data is just as relevant to listening or relay stations and this data can can be saved by them for re-transmit purposes such as store and forward type actions later.

During an active message transmission session there are additional options that can be used by both sending and receiving stations that can potentially be very usefull. Two possible options are detailed below:

1) The receive station can opt to include a pre-message on the front of any ACK or NACK messages sent to the sending station. An example of this would be the receive station sending out a PEND pre-message with details (in stub message format) of a message waiting to be sent from its outbox message queue. This information is delivered to the group and can be used later by those group members to retrieve any messages of interest by using the message pull techniques described earlier.

2) The sending station is readying to go QRT following completion of the current message transfer. Sending station can opt to send out a POST pre-message on the front of any transmission that goes to the group. The POST pre-message may contain a message such as ‘GOING QRT AFTER XFER COMPLETE 73’. The POST messages can be delivered to the group ***during*** an active message transfer session.

Additional options for repeating critical fragments or the entire message are as follows:

1) Repeating critical fragments. Up to the first three or so fragments can be repeated at various points throughout the message to increase the probability that these fragments are communicated successfully with the first message send. These first few fragments are usually sufficient to cover the message ID and the send list, depending on fragmentation size, and represent key information allowing other stations to identify and request those messages subsequently.

2) Repeating the entire message multiple times. Second and Third copies of the message can be interlaced with the original message so that diffent fragments occupy similar positions in the sequence of communication. For example, one interlace can sequence forward through the fragments while the other interlace can start from the end and sequence backwards. Another example is a type of phase shift where the message is repeated at one third distance along the message and two thirds distance along the message by interlacing these with the original message. By doing this, intermittent impacts to the communication stream may not necessarily cause the fragments to fail as they appear at multiple other positions along the message stream timewise. Typically this may be used with some of the higher speed modes of communication.

i. Transport Layer Commands

Format	Description	Formats
ACK	Acknowledgment from a receive station that message transferred successfully	ACK WH6ABC
ACK?	Query to a receive station to ask if the station received the message successfully.	WH6GHI ACK? WH6ABC
KCAN	Negative acknowledgement sent by a receive station notifying the sender of missing fragments for subsequent re-transmit	KCAN (F3-16) WH6GHI KCAN +[28[ACFGH WH6GHI

ii. Message Acknowledgement

After the message is sent, the receiving stations send a reply back to denote if the message was received successfully or not. If the active session is being done on a single channel, as is often the case with SAAMFRAM General format, then the responses are sent back one by one in a sequential manner under the control of the sending station. Whereas with SAAMFRAM – JS8 the responses can be sent back in a sequenctial manner via a single channel or they can be sent back in parallel via multiple channels as JS8 / JS8Call has capabilities to receive multiple channels simultaneously. Processing of re-transmits will need to be done in a sequential manner as the sending station must transmit on one channel only, however the sending station can listen to and merge all of the NACK message requests to create a single re-transmit that contains all of the missing fragments for all of the stations that replied NACK thereby potentially reducing the number of back and forth re-transmit attempts between the stations.

- ACK. This message is sent by the receiving station to denote that all fragmants have been received and verified. Successfully. Denotes successful transmission of the message
- KCAN – General Format. This is NACK spelled backwards. During testing with higher speed modes, frequently a NACK message would appear as an ACK. So to reduce the possibility of this being decoded as an ACK by mistake, the text was spelled backwards. KCAN (F1,F2) dontes fragments F1 and F2 failed. In SAAMFRAM, the receiving stations are given a varying amount of time to respond with the relevant ACK/NACK code that can be up to 12 seconds depending on the mode. If the sending station does not receive a reply in this timeframe, the sending station will send out a query to prompt the station for status. If the station fails to respond after several attempts, the sender moves to the next station on the list.
- KCAN - JS8 Format. This is the JS8 Format NACK code. With JS8, there are two different ways to send a nack code and the one that is the shortest length is the one chosen for use. One describes the fragments that were received while the other describes fragments that are missing. KCAN +[3A denotes that only fragments 3 thru A were received all others are missing. If the last fragment, which contains the total number of fragments, was not received, then the only choice is to send back which fragments were received as there is no way to calculate which fragments are missing. KCAN -[8B denotes that the last fragmant and all other fragments except 8 thru B were received. KCAN +[257 denotes that fragments 2 thru 5 and 7 were received only. KCAN +[28[ACFGH denotes fragments 2 thru 8, A thru C and F, G, H were received. KCAN -[26[9CGH denotes fragments 2 thru 6, 9 thru C, G, H are missing. -- denotes no frames received.
- ACK? This message is sent by the sending station to query the ACK status of the receiving station.After sending the ACK? message, the receiving station is given a varying amount of time to respond depending on the digital mode selected. This can be up to 12 seconds. This is repeated until either the receiving station responds or a preset number of retries is completed.If the receiving station sends back a NACK <missing fragments> reply then the sending station will immediately re-transmit the named fragments. This process will continue until either the receiving station sends back an ACK message or a preset number of resend attempts has been completed;The sending station will cycle back and forth in this manner several times but in the event that the receiving station is simply unable to copy the sent message fragments, after a present number of resend attempts, the resend process to that station will be aborted and the process moved on to the next station on the list if any.

II. CRC Checksums

Selecting which CRC polynomials to use is not a trivial task. The wide variety of digital modes as well as the diverse band propagation conditions present some challenges to the selection of polynomials.

The general purpose polynomials used are derived from the Best CRC Polynomials document of Philip Koopman, Carnegie Mellon University published under Creative Commons License: https://creativecommons.org/licenses/by/4.0/.

These polynomials assume a low constant random independent bit error rate (BER) which may be appropriate for some combinations of mode and band condition but not others. Testing so far has indicated that these polynomials work well for the purpose but may be revised in the future as more testing is completed and more information becomes available. Please note that the SAAMFRAM general mode has a two layer CRC check; the first to validate the individual fragments and the second to validate the entire reconstituted message. The SAAMFRAM JS8 format utilizes a full message CRC check only on the fully reconstituted message. This format relies on the underlying JS8 protocol for reliable delivery of the message with the dropped frame indication from JS8Call being the indicator for wheather a fragment re-transmit is necessary for the impacted fragment.

There are four polynomials used to protect / verify the data being transferred. The CRC logic uses base 32 as this can be efficiently represented using a single alpha numeric character 0-9 and A-V. Each charactrer of the CRC checksum therefore represents 5 bits. The number of characters in a given send string is used to determine which polynomial to use based on a worst case character size of 8 bits (some digital communication modes use less than 8 bits to represent each character. Notwithstanding this 8 bits per character is assumed as a general standard in this regard for use across many digital modes)

The specific polynomials used are:

- Polynomial 0x247. This polynomial has a 10 bit CRC size (2 digits) to protect data upto a maximum length of 501 bits or 62 x 8 bit characters at Hamming Distance HD=4
- Polynomial 0x327. This polynomial has a 10 bit CRC size (2 digits) to protect data upto a maximum length of 1013 bits or 126 x 8 bit characters at Hamming Distance HD=3
- Polynomial 0x4306. This polynomial has a 15 bit CRC size (3 digits) to protect data upto a maximum length of 16368 bits or 2046 x 8 bit characters at Hamming Distance HD=4
- Polynomial 0xc1acf. This polynomial has a 20 bit CRC size (4 digits) to protect data upto a maximum length of 524267 bits or 65533 x 8 bit characters at Hamming Distance HD=4

lll. Requesting additional checksums

After the active message transfer process is complete and all the fragments received, the message should pass all checksum tests. If for some reason the message appears to be corrupted even after doing this, it is possible to request an alternate set of checksums using the application layer commands. These can then be used to identify if somehow an invalid fragment managed to pass the checksum test in error and to then request a re-transmit of that fragment. The probability of this happening is relatively small however it is not zero. The functionality to request alternative CRCs using different polynomials is there in case it is needed.

IV. Fragmentation

This is best illustrated with examples. The following sections contain examples of SAAMFRAM communications showing the full text of a message exactly as it is sent/received. For illustration purposes only, the examples use a fragmentation size of 10 characters. In reality a larger fragment size would typically be used. The SAAM-MAIL application reference platform can fragment (prior to TX) and defragment (following RX) these messages and display the full ICS form with the fields populated as per the transmission.

Fragment Length

The critical message utilizes fragmentation to divide the message up into segments. The station operator decides the fragmentation size to be used for this process. Typically fragment sizes may vary from as small as 10 characters per fragment to 200 characters per fragment. Fragment length is chosen depending on the different band conditions as well as the different digital comuunication modes. For example JS8 mode would work well with longer fragment lengths as JS8 already has frame processing, CRC checks and Forward Error Correction (FEC) built in. Higher speed fldigi modes may benefit from shorter fragment lengths. Also poor band conditions may necessitate using shorter fragment lengths as longer length fragments may require additional retransmits. The length of all fragments is specified by the station operator and incorporated near the start of the message and in the general format also on each start fragment tag. All fragments except the last fragment will be of this length. The last fragment will be the length of the ramaining message. The last fragment also contains a four digit full message checksum at the end of the message which is used to validate the entire message once each of the fragments is reconstituted into the defragmented original message form.

V. Fragmentation - General Format

i. Recognizing and Reading Content - General Format

Fragmentation process:

Each fragment starts with a fragment number and number of fragments i.e. [F1,8] would represent Fragment 1 in an 8 fragment sequence. Each fragment ends with a checksum for the fragment i.e. [G8] would represent a 10 bit (2 x 5 bit base 32 characters) CRC checksum of G8. This is the checksum of the content contained between the start fragment tag and checksum fragment tag bookends.

The individual fragments are numbered sequentially starting with 1. Each fragment starts with a start fragment tag. For example [F1,16]…..[F2,16]….[F3,16]...[F16,16]

Each fragment ends with a checksum. The last frament also has EOM for end of message. for exmaple [U5]...[52]...[EL]...[G8]EOM

so excluding the data content (represented with …) these tags read as follows [F1,16]...[U5][F2,16]...[52][F3,16]...[EL]...[F16,16]...[G8]EOM

The last fragment has a 4 digit full message checksum appended to the end of the message and protected with the fragment checksum followed by end of message (EOM) and the sender callsign. The last fragment may be shorter than the specified fragment size. for example…

….74CI[G8]EOM WH6KLM

so excluding content the message reads as follows content is replaced with ‘...’

WH6KLM: @HINET BOS [F1,16]...[U5][F2,16]...[52][F3,16]...[EL][F4,16]...[MA][F5,16]...[HO][F6,16]...[ET][F7,16]...[2R][F8,16]...[GA][F9,16]...[R1][F10,16]...[88][F11,16]...[T0][F12,16]...[QS][F13,16]...[T4][F14,16]...[RL][F15,16]...[QM][F16,16]...[G8]EOM WH6KLM

A complete message as transmitted over fldigi therefore looks as follows.

WH6KLM: @HINET BOS [F1,16]{DATA~750c[U5][F2,16]deca_3731a[52][F3,16]4b5~WH6GGO[EL][F4,16]~~10~This [MA][F5,16]is a test [HO][F6,16]message~IC[ET][F7,16]S 214~1.3~[2R][F8,16]My Test In[GA][F9,16]cident~1~0[R1][F10,16]9-19-2022~[88][F11,16]09-20-2022[T0][F12,16]~2300~2300[QS][F13,16]~Lawrence~[T4][F14,16]Puna QTH~O[RL][F15,16]perator/45[QM][F16,16]~}74CI[G8]EOM WH6KLM

Defragmentation Process:

After removing the start frame tag, the checksum tag, and the EOM and end of message checksum, we are left with the content which can be utilized by the presentation layer as follows:

WH6KLM: @HINET BOS {DATA~750cdeca_3731a4b5~WH6GGO~~10~This is a test message~ICS 214~1.3~My Test Incident~1~09-19-2022~09-20-2022~2300~2300~Lawrence~Puna QTH~Operator/45~} WH6KLM

The end result is human readable data representing content to be filled into the fields on the ICS 214 form sequentially. The data is in a format not unlike the comma separated value or CSV format found in many mainstream speadsheet and other applications, the only difference being that instead of having a comma to separate the values, SAAMFRAM uses the tilde ~ separator to separate the individual field values. As can be seen at the very end of the message there is a /45~ which is run length encoding meaning replace this with 45 ~ characters. This is to be read as each of these 45 fields are empty and contain no content. So, reading the content as follows from top left to bottom right: Sender callsign, to group name, beginning of session indicator ‘BOS’, start of critical content DATA ‘{DATA’, message id, receipient call sign list, priority, fragment size, message subject, form template name, form template version, individual field data: ‘My Test Incident~1~09-19-2022~09-20-2022~2300~2300~Lawrence~Puna QTH~Operator/45~’ , end of critical message designator ‘}’, sender callsign.

ii. Complete Examples – General Format

Some examples of full messages as they are sent to and received from fldigi using the general format are as follows:

ICS 214 Content only:

WH6KLM: @HINET BOS [F1,16]{DATA~750c[U5][F2,16]deca_3731a[52][F3,16]4b5~WH6GGO[EL][F4,16]~~10~This [MA][F5,16]is a test [HO][F6,16]message~IC[ET][F7,16]S 214~1.3~[2R][F8,16]My Test In[GA][F9,16]cident~1~0[R1][F10,16]9-19-2022~[88][F11,16]09-20-2022[T0][F12,16]~2300~2300[QS][F13,16]~Lawrence~[T4][F14,16]Puna QTH~O[RL][F15,16]perator/45[QM][F16,16]~}74CI[G8]EOM WH6KLM

ICS 214 Content + Pre-message:

WH6KLM: @HINET BOS PEND(750cdeca_37168699,wh6ggo,QV)[F1,16]{DATA~750c[U5][F2,16]deca_3731a[52][F3,16]4b5~WH6GGO[EL][F4,16]~~10~This [MA][F5,16]is a test [HO][F6,16]message~IC[ET][F7,16]S 214~1.3~[2R][F8,16]My Test In[GA][F9,16]cident~1~0[R1][F10,16]9-19-2022~[88][F11,16]09-20-2022[T0][F12,16]~2300~2300[QS][F13,16]~Lawrence~[T4][F14,16]Puna QTH~O[RL][F15,16]perator/45[QM][F16,16]~}74CI[G8]EOM WH6KLM

ICS 214 Content + ICS 214 Form Template:

WH6KLM: @HINET BOS [F1,35]{DATA~750c[U5][F2,35]deca_3731a[52][F3,35]4b5~WH6GGO[EL][F4,35]~~10~This [MA][F5,35]is a test [HO][F6,35]message~IC[ET][F7,35]S 214~1.3~[2R][F8,35]My Test In[GA][F9,35]cident~1~0[R1][F10,35]9-19-2022~[88][F11,35]09-20-2022[T0][F12,35]~2300~2300[QS][F13,35]~Lawrence~[T4][F14,35]Puna QTH~O[RL][F15,35]perator/45[QM][F16,35]~}{FORM~IC[L6][F17,35]S~v1.0~ICS[77][F18,35] 214 Form~[HS][F19,35]0B,AI~02~0[J3][F20,35]6~I1~42~00[93][F21,35]~06~P2~10~[R6][F22,35]06~O2~D4,1[2O][F23,35]3,@1,D5,13[2D][F24,35]~T7,13,@1,[D8][F25,35]T8,13~01~0[7K][F26,35]2~06~N5,42[OK][F27,35]~H5,42~00~[RB][F28,35]06~IO,42~0[C8][F29,35]1~05~09,R4[BM][F30,35]~N5,IO,H5~[I6][F31,35]@I,0F~22,2[18][F32,35]2,26~05~09[IL][F33,35],AH~D3,NB~[KU][F34,35]@K,0F~22,3[OH][F35,35]2}1794[8Q]EOM WH6KLM

VI. Fragmentation – JS8 Optimized Format

i. Recognizing and Reading Content - JS8 Optimized Format

This form of fragmentation is specially optimized for JS8 only modes. Messages can be up to 36 fragments in length and have a variable length fragment size. Square brackets followed by a single character mark the start of a frame. The character represents a base 36 number and uses numerals 0-9 and letter A thru Z. The fragments are number sequentially starting with 0.

The start frame tag of each message fragment:

[0...[1…[2...[3...[4...[F...[GNG43/E

The last fragment number is also the same as the total number of fragments in the message. The last fragment also contains a 4 character base 32 checksum (20 bits) and the end message designator /E

During re-transmit of any missing frames, /E is used to denote the end of the retransmit of frames. Stations will reply back only when they hear /E at the end of the initial message send or on the end of the re-transmitted fragments.

The frame verification technique uses the JS8/JS8Call built in missing frame detection to determine if the frame was transmitted successfully or not. One or more missing frames indicate that the fragment is to be discarded.

This format keeps the overhead ratio of tags (for fragment identification + verification) to content to a minimum. Typically 2 characters per fragment + additional 6 for the message checksum and end of message indicator.

A turbo mode 35 character JS8 fragment would normally take 5 JS8 frames but would now take an additional 2 characters for every frame and an additional 6 at the end. So 5 fragments of 35 characters each would be 25 frames normally and after fragmentation be 27 frames total or about a 7% overhead

ii. Complete Examples – JS8 Optimized Format

Content only message

WH6KLM: @HINET BOS [0{DATA~750c[1deca_3731a[24b5~WH6GGO[3~~10~This [4is a test [5message~IC[6S 214~1.3~[7My Test In[8cident~1~0[99-19-2022~[A09-20-2022[B~2300~2300[C~Lawrence~[DPuna QTH~O[Eperator/45[F~} [GNG43/E WH6KLM

Content + Template

WH6KLM: @HINET BOS [0{DATA~750c[1deca_3731a[24b5~WH6GGO[3~~10~This [4is a test [5message~IC[6S 214~1.3~[7My Test In[8cident~1~0[99-19-2022~[A09-20-2022[B~2300~2300[C~Lawrence~[DPuna QTH~O[Eperator/45[F~} {FORM~IC[GS~v1.0~ICS[H 214 Form~[I0B,AI~02~0[J6~I1~42~00[K~06~P2~10~[L06~O2~D4,1[M3,@1,D5,13[N~T7,13,@1, [OT8,13~01~0[P2~06~N5,42[Q~H5,42~00~[R06~IO,42~0[S1~05~09,R4[T~N5,IO,H5~[U@I,0F~22,2[V2,26~05~09[W,AH~D3,NB~[X@K,0F~22,3[Y2}[ZUU4U/E WH6KLM

Reading the content of a JS8 message can be done using a similar technique to that already described for the General format. The difference being the formats of the frame tags and end of message tag are slightly different as described above.

8. Appendix A - Full ICS TAG Reference.

```
field_names = {

  ##################################
  # sequence of '0' control codes #
  ##################################

  #""" column control codes """
  '00'    : 'NewColumn',
  '01'    : 'EndColumn',
  '02'    : 'StartColumn',

  #""" separator control code """
  '03'    : '10xSeparator',
  '04'    : '15xSeparator',
  '05'    : '20xSeparator',
  '06'    : '30xSeparator',
  '07'    : '60xSeparator',
  '08'    : '80xSeparator',

  #""" sub heading control code """
  '09'    : 'SubHeading',

  #""" combo control code """
  '0A'    : 'Combo',

  #""" main heading control code """
  '0B'    : 'MainHeading',

  #""" repeat next field x number of times"""
  '0C'    : '2xField Repeat',
  '0D'    : '3xField Repeat',
  '0E'    : '4xField Repeat',

  #""" align line field widths. (anchor line,adjustment line) relative values """
  '0F'    : 'LineAlign 1,-1',

  #""" add a filler field that spans number of filelds on relative line index """
  '0G'    : 'FillerSpan 5,-1',
  '0H'    : 'FillerSpan 5,1',
  '0I'    : 'LineAlign 1,-2',
  '0J'    : 'LineAlign 1,-3',
```

```
#""" add a filler field that spans number of filelds on relative line index """
'0K'    : 'FillerSpan 1,-1',
'0L'    : 'FillerSpan 2,-1',
'0M'    : 'FillerSpan 3,-1',
'0N'    : 'FillerSpan 4,-1',

#""" add a filler field that spans number of filelds on relative line index """
'0O'    : 'FillerSpan 1,1',
'0P'    : 'FillerSpan 2,1',
'0Q'    : 'FillerSpan 3,1',
'0R'    : 'FillerSpan 4,1',

#""" radio button control code """
'0S'    : 'RadioButton',

#""" checkbox control code """
'0T'    : 'Checkbox',

#""" align field with prev row field# 6 """
'0U'    : 'FillerPad 1',
'0V'    : 'FillerPad 2',
'0W'    : 'FillerPad 3',
'0X'    : 'FillerPad 4',
'0Y'    : 'FillerPad 5',
'0Z'    : 'FillerPad 6',

##################################
# sequence of '@' control codes #
##################################
'@1'    : 'Spacer 5',
'@2'    : 'Spacer 10',
'@3'    : 'Spacer 15',
'@4'    : 'Spacer 20',
'@5'    : 'Spacer 25',
'@6'    : 'Spacer 30',
'@7'    : 'Spacer 35',
'@8'    : 'Spacer 40',
'@9'    : 'Spacer 45',
'@A'    : 'Spacer 50',
'@B'    : 'Spacer 55',
'@C'    : 'Spacer 60',
'@D'    : 'Spacer 65',
'@E'    : 'Spacer 70',
'@F'    : 'Spacer 75',
'@G'    : 'Spacer 80',
```

```
'@H'    : '3xLineRepeat',
'@I'    : '5xLineRepeat',
'@J'    : '10xLineRepeat',
'@K'    : '15xLineRepeat',
'@L'    : '20xLineRepeat',
'@M'    : '30xLineRepeat',
'@N'    : '40xLineRepeat',
'@O'    : '50xLineRepeat',

#""" Radio button group """
'@Q'    : 'RadioBtnGroup',
#""" option menu """
'@R'    : 'OptionMenu',
#""" option menu """
'@S'    : 'CkBoxGroup',

########################################
# sequence of numeric codes            #
# entry fields + dynamic content macros #
########################################
'1A'    : '13x1 %CALLSIGN%',
'1B'    : '35x1 %OPERATORNAME%',
'1C'    : '23x1 %DATETIME%',
'1D'    : '10x1 %DATE%',
'1E'    : '10x1 %TIME%',
'1F'    : '30x1 %INCIDENTNAME%',
'1G'    : '30x1 %OPERATORTITLE%',
'1'     : '1x1',
'2'     : '2x1',
'3'     : '3x1',
'4'     : '4x1',
'5'     : '5x1',
'6'     : '6x1',
'7'     : '7x1',
'8'     : '8x1',
'9'     : '9x1',
'10'    : '10x1',
'11'    : '11x1',
'12'    : '12x1',
'13'    : '13x1',
'14'    : '14x1',
'15'    : '15x1',
'16'    : '16x1',
'17'    : '17x1',
'18'    : '18x1',
```

```
'19'    : '19x1',
'20'    : '20x1',
'21'    : '25x1',
'22'    : '30x1',
'23'    : '35x1',
'24'    : '40x1',
'25'    : '45x1',
'26'    : '50x1',
'27'    : '55x1',
'28'    : '60x1',
'29'    : '65x1',
'30'    : '70x1',
'31'    : '75x1',
'32'    : '80x1',
'33'    : '60x10',
'34'    : '70x10',
'35'    : '80x10',
'36'    : '90x10',
'37'    : '91x10',
'38'    : '92x10',
'39'    : '93x10',
'40'    : '94x10',
'41'    : '90x5',
'42'    : '0x1',
'43'    : '0x3',
'44'    : '0x5',
'45'    : '0x10',
'46'    : '0x15',
'47'    : '0x20',

########################################
# text field tags                      #
########################################
'A1'    : 'Approved by',
'A2'    : 'Assignment',
'A3'    : 'Approved by (CUL)',
'A4'    : 'Assignment List (ICS 204)',
'A5'    : 'Add',
'A6'    : 'Approved by Incident Commander',
'A7'    : 'Agency / Organization Representatives',
'A8'    : 'Agency / Organization',
'A9'    : 'Air Ops Branch Director',
'AA'    : 'Ambulance Services',
'AB'    : 'Address and Phone',
'AC'    : 'ALS',
'AD'    : 'Address Latitude and Longitude if Helipad',
```

```
'AE'    : 'Air',
'AF'    : 'Approved by (Safety Officer)',
'AG'    : 'Assignment / Location',
'AH'    : 'Activity Log',
'AI'    : 'Activity Log (ICS 214)',

'B1'    : 'Basic Radio Channel Use',
'B2'    : 'Branch',
'B3'    : 'Branch Director',

#""" spacer text field """
'B4'    : '     ',

#""" vertical separator text field """
'B5'    : '|',
'B6'    : 'BLS',
'B7'    : 'Burn Center',

'C1'    : 'Call Sign',
'C2'    : 'Ch#',
'C3'    : 'Channel Name',
'C4'    : 'COMMUNICATIONS LOG (ICS 309)',
'C5'    : 'Contact e.g. Phone / Pager / RF',
'C6'    : 'Communications (radio and/or phone contact numbers needed for this assignment)',
'C7'    : 'Chief',
'C8'    : 'Communications Unit',
'C9'    : 'Cost Unit',
'CA'    : 'Comp / Claims Unit',
'CB'    : 'Contact Number(s) / Frequency',
'CC'    : 'Cost',
'CD'    : 'Communications Resource Availability Worksheet (ICS 217A)',
'CE'    : 'Channel Configuration',
'CF'    : 'Channel Name / Trunked Radio System Talkgroup',

'D1'    : 'Date',
'D2'    : 'Date / Time Prepared',
'D3'    : 'Date / Time',
'D4'    : 'Date From',
'D5'    : 'Date To',
'D6'    : 'Division',
'D7'    : 'Division / Group Supervisor',
'D8'    : 'Deputy',
'D9'    : 'Division / Group',
'DA'    : 'Documentation Unit',
'DB'    : 'Demobilization Unit',
'DC'    : 'Director',
```

```
    'DD'    : 'Detailed Item Description, Vital Characteristics, Brand, Specs, Experience, Size,
etc.',
    'DE'    : 'Delivery / Reporting Location',
    'DF'    : 'Description',
    'DG'    : 'Date / Time (Optional)',

    'E1'    : 'Express Sender',
    'E2'    : 'Estimated',
    'E3'    : 'Eligible Users',

    'F1'    : 'From',
    'F2'    : 'For Operational Period',
    'F3'    : 'Function',
    'F4'    : 'From (Name / Position)',
    'F5'    : 'Facilities Unit',
    'F6'    : 'Food Unit',
    'F7'    : 'Finance / Administration Section',
    'F8'    : 'Finance',
    'F9'    : 'Finance Section Chief Name',
    'FA'    : 'Frequency Band',

    'G1'    : 'Group',
    'G2'    : 'General Situational Awareness',
    'G3'    : 'Ground Support Unit',
    'G4'    : 'Ground',
    'G5'    : 'General Message (ICS 213)',

    'H1'    : 'Hospitals',
    'H2'    : 'Hospital Name',
    'H3'    : 'Helipad',
    'H4'    : 'Hazards / Risks',
    'H5'    : 'Home Agency and Unit',

    'I1'    : 'Incident Name',
    'I2'    : 'IAP Page',
    'I3'    : 'INCIDENT RADIO COMMUNICATIONS PLAN (ICS 205)',
    'I4'    : 'Information',
    'I5'    : 'Incident Objectives (ICS 202)',
    'I6'    : 'Incident Actiona Plan (the items checked below are included in this Incident Action
Plan)',
    'I7'    : 'ICS 203',
    'I8'    : 'ICS 204',
    'I9'    : 'ICS 205',
    'IA'    : 'Indicate cell, pager, or radio (frequency/system/channel)',
    'IB'    : 'ICS 205A',
    'IC'    : 'ICS 206',
    'ID'    : 'ICS 207',
```

```
'IE'    : 'ICS 208',
'IF'    : 'ICS 202',
'IG'    : 'Incident Commander(s) and Command Staff',
'IH'    : 'IC / UCs',
'II'    : 'Item Description',
'IJ'    : 'Incident Action Safety Analysis (ICS 215A)',
'IK'    : 'Incident Number',
'IL'    : 'Incident Area',
'IM'    : 'Individual Activity Log (ICS 214A)',
'IN'    : 'ICS Section',
'IO'    : 'ICS Position',

'K1'    : 'Kind',

'L1'    : 'Leader',
'L2'    : 'Liason Officer',
'L3'    : 'Location',
'L4'    : 'Level of Service',
'L5'    : 'Level',
'L6'    : 'Low',
'L7'    : 'Logistics',
'L8'    : 'Logistics Order Number',

'M1'    : 'Message',
'M2'    : 'Mode (A,D,M)',
'M3'    : 'Map / Chart',
'M4'    : 'Medical Unit',
'M5'    : 'Medical Plan',
'M6'    : 'Medical Aid Stations',
'M7'    : 'Medical Emergency Procedures (Be Brief)',
'M8'    : 'Medical Plan (ICS 206)',
'M9'    : 'Mitigations',
'MA'    : 'Major Events',

'N1'    : 'Name / Contact Number(s)',
'N2'    : '# or Persons',
'N3'    : 'Notes',
'N4'    : 'Name / Function',
'N5'    : 'Name',
'N6'    : 'No',
'N7'    : 'New Status',
'N8'    : 'Needed Date / Time (local 24 hr)',
'N9'    : 'Name of Supplier',
'NA'    : 'Name of Auth Logistics Rep',
'NB'    : 'Notable Activities',
```

```
'O1'    : 'Operator Name',
'O2'    : 'Operational Period',
'O3'    : 'Operations Personnel',
'O4'    : 'Operations Section Chief',
'O5'    : 'Objectives',
'O6'    : 'Operational Period Command Emphasis',
'O7'    : 'Other Attachments',
'O8'    : 'Organizational Assignment List (ICS 203)',
'O9'    : 'Operations Section',
'OA'    : 'Order',
'OB'    : 'Order was Requested by',

'P1'    : 'Position / Title',
'P2'    : 'Page #',
'P3'    : 'Primary Contact',
'P4'    : 'Prepared by',
'P5'    : 'Planning Section',
'P6'    : 'Public Info Officer',
'P7'    : 'Procurement Unit',
'P8'    : 'Paramedics',
'P9'    : 'Prepared by (MUL)',
'PA'    : 'Priority',
'PB'    : 'Point of Contact',
'PC'    : 'Prepared by (Safety Officer)',
'PD'    : 'Prepared by (Operations Section Chief)',

'Q1'    : 'Qty',

'R1'    : 'RX Freq',
'R2'    : 'RX Tone',
'R3'    : 'Remarks',
'R4'    : 'Resources Assigned',
'R5'    : 'Reporting Location',
'R6'    : 'Resource Unit',
'R7'    : 'Resource Status Change (ICS 210)',
'R8'    : 'Resource #',
'R9'    : 'Resource Request Message (ICS 213 RR)',
'RA'    : 'Resource Request Number',
'RB'    : 'Requester',
'RC'    : 'Requested',
'RD'    : 'Requested by Name / Position',
'RE'    : 'Routine',
'RF'    : 'Reply / Comments from Finance',
'RG'    : 'RX Freq N/W',
'RH'    : 'RX Tone/NAC',
```

```
'S1'    : 'Subject',
'S2'    : 'Station ID',
'S3'    : 'Special Instructions (Be Brief)',
'S4'    : 'Staging Area',
'S5'    : 'Special Equipment and Supplies',
'S6'    : 'Special Instructions',
'S7'    : 'Safety Plan Required',
'S8'    : 'Safety Plan(s) Located at',
'S9'    : 'Signature',
'SA'    : 'Staging Area',
'SB'    : 'Safety Officer',
'SC'    : 'Situation Unit',
'SD'    : 'Supply Unit',
'SE'    : 'Safety Message / Plan (ICS 208)',
'SF'    : 'Safety Message / Expanded Safety Message, Safety Plan, Site Safety Plan',
'SG'    : 'Substitutes and / or Suggested Sources',
'SH'    : 'Section Chief Name for Approval',
'SI'    : 'Supplier Phone / Fax / Email',

'T1'    : 'To (Name/Position)',
'T2'    : 'Time',
'T3'    : 'Task',
'T4'    : 'Task Name',
'T5'    : "Track and Increment your page #'s (Default is 1)",
'T6'    : 'To',
'T7'    : 'Time From',
'T8'    : 'Time To',
'T9'    : 'TX Freq',
'TA'    : 'TX Tone',
'TB'    : 'Task #',
'TC'    : 'Technical Specialists',
'TD'    : 'Time Unit',
'TE'    : 'Transportation',
'TF'    : 'Travel Time',
'TG'    : 'Trauma Center',
'TH'    : 'Time and Date of Change',
'TI'    : 'Type',
'TJ'    : 'TX Freq N/W',
'TK'    : 'TX Tone/NAC',
'TL'    : 'Test1,Test2,Test3,test4',

'U1'    : 'Urgent',

'W1'    : 'Work Assignments',
'W2'    : 'Weather forecast / Tides / Currents',
'W3'    : 'Worksheet Incident or Event Name',
```

```
  'Y1'    : 'Yes',
  'Y2'    : 'Yes,No',

  'Z1'    : 'Zone Grp',

}
```

9. Appendix B – Message ID Algorithm

```
  """ This method creates a unique ID based on the callsign and the month, day, hour, minute,
second"""
  def getEncodeUniqueId(self, callsign):
    sys.stdout.write("getEncodeUniqueId\n")

    chars = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    charsLen = len(chars)
    num = 0
    for i, c in enumerate(reversed(callsign.upper())):
      num += chars.index(c) * (charsLen ** i)
    ID = '{:02x}'.format(num) + '_' + '{:02x}'.format(int(datetime.utcnow().strftime('%m%d%H%M%S')))
    return (ID)



  """ This method decodes the callsign from the ID string"""
  def getDecodeCallsignFromUniqueId(self, ID):
    """ use the following to reverse the callsign from the ID string to show who created the
email"""
    chars = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    charsLen = len(chars)
    hexnum = '0x' + ID.split('_',1)[0]

    timestamp_string = ID.split('_',1)[1]
    inttime = int(timestamp_string,16)
    sys.stdout.write("reverse encoded timestamp is: " + str(inttime) + "\n")
    sys.stdout.write("seconds = " + str(inttime % 100) + "\n")
    sys.stdout.write("minutes = " + str((inttime / 100) % 100) + "\n")
    sys.stdout.write("hours   = " + str((inttime / 10000) % 100) + "\n")
    sys.stdout.write("days    = " + str((inttime / 1000000) % 100) + "\n")
    sys.stdout.write("month   = " + str((inttime / 100000000) % 100) + "\n")

    intnum = int(hexnum,16)
    callsign = ""
    while intnum:
      callsign = chars[intnum % charsLen] + callsign
      intnum //= charsLen
    return callsign
```

```
  """ This method decodes the callsign from the ID string"""
  def getDecodeTimestampFromUniqueId(self, ID):
    """ use the following to reverse the callsign from the ID string to show who created the email"""
    chars = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    charsLen = len(chars)

    timestamp_string = ID.split('_',1)[1]
    inttime = int(timestamp_string,16)

    time_seconds = int(inttime % 100)
    time_minutes = int((inttime / 100) % 100)
    time_hours   = int((inttime / 10000) % 100)
    time_days    = int((inttime / 1000000) % 100)
    time_month   = int((inttime / 100000000) % 100)

    timestamp = calendar.month_abbr[time_month] + " {day:02d} at {hours:02d}:{minutes:02d}:{seconds:02d}".format(day=time_days,

hours=time_hours, minutes=time_minutes,

seconds=time_seconds )
    sys.stdout.write("reverse encoded timestamp is: " + timestamp + "\n")

    return timestamp
```