Section 11
ETMS Electronic Mail Function

The *EMail* function constructs and sends messages to the Aeronautical Radio Incorporated (ARINC), National Airspace Data Interchange Network (NADIN) or another ETMS workstation. There are two types of messages: advisory and general. A message can be constructed in free format, or prestored messages can be selected. See Figure 11-1.

NOTE: In this section, inputs, outputs, and processing are not described for any *EMail* routines that have minimal processing. Also, if a routine interface is listed in the Routine Interface section but not mentioned in the Processing section, it is because that interface is used numerous times; only major routines are mentioned in the Processing section.

The *ETMS Electronic Mail* (*EMail*) is a single process function consists of the following two modules:

- *email_p* module
- *email_new_p* module

Execution Control

EMail is initiated by selecting the *EMail* icon on the *Tool Manager (TMGR)* bar. *TMGR* invokes a script named **start_email** in the **/etms5/com** directory. The script invokes an executable file from the **/etms5/email** directory named **email**.

Input

Email reads the following files as input:

- **arinc** contains the ARINC addresses that can be selected in the addresses popup on the *Email* interface. The **arinc** file is located in the **/email/data/address_files** directory.
- **groups_to_list** contains a list of group names that the user wishes to display in the groups popup. If the file is empty or does not exist, a warning is displayed at startup and all groups contained in the **/shared/data/address_groups** directory are displayed. The **groups_to_list**

file also has an **ignore_this_file** option; if the **ignore_this_file** option is the first string in the file, the file is ignored and a warning is not displayed.

- **email_function_enable_table** contains data strings that are responsible for enabling *EMail* to construct and send advisory messages and send any type of message to the NADIN and ARINC circuits. The **email_function_enable_table** is located in the **/email/config** directory.

- **etms** contains the ETMS workstation addresses that can be selected in the addresses popup on the *EMail* interface. The **etms** file is located in the **/email/data/address_files** directory.

- **nadin** contains the NADIN addresses that can be selected in the addresses popup on the *EMail* interface. The **nadin** file is located in the **/email/data/address_files** directory.

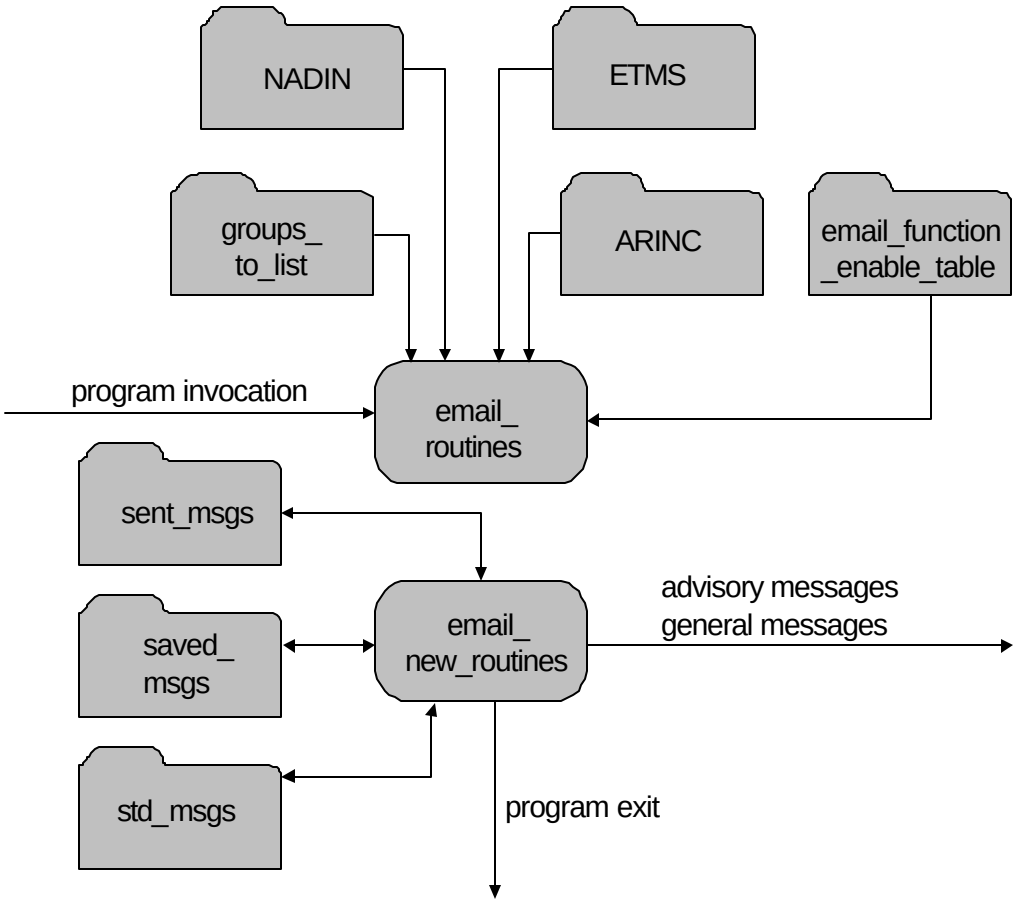

Figure11-1. Data Flow of the EMail Process

ETMS directory parameter - *EMail* requires the root directory to build pathnames. To obtain this information, the *get_etms_paths* routine is used. The routine first looks for a **- path** command line argument followed by the directory name. The second attempt is to read the directory from a file named ETMSPATH. If ETMSPATH does not exist, the routine will set a default value of "/**etms**".

Saved Messages - Messages saved from the message pad on the *EMail* interface. These messages are saved by selecting the Save Message button. The messages are stored in the **/email/data/saved_msgs** directory.

Sent Messages - Messages sent from *EMail*. When a message is sent, it is date and time stamped and saved in the **/email/data/sent_msgs** directory. Types of messages sent from *EMail* are either general or advisory messages destined for an ETMS workstation, the NADIN circuit, or the ARINC circuit.

Standard Messages - Messages that are commonly used prestored templates. These messages are stored in the **/email/data/std_msgs** directory.

Output

Saved Messages - Messages saved from the message pad on the *EMail* interface. These messages are saved by selecting the Save Message button and are stored in the **/email/data/saved_msgs** directory.

Sent Messages - Messages sent from *EMail*. When a message is sent, it is date and time stamped and saved in the **/email/data/sent_msgs** directory. Types of messages sent from *EMail* are either general or advisory messages destined for an ETMS workstation, the NADIN circuit, or the ARINC circuit.

Design Issues

The *EMail* function utilizes Network Addressing Toolkit calls (**net$_**) to communicate with the *Router (RTR)* process. The *RTR* is used by *EMail* to transmit messages through ETMS communications. As a result, *RTR* must run on every workstation used to send or receive messages. The ETMS communications class identification is email, which is used by the *site.sw* program to determine what process is sending/receiving messages. The graphical user interface (GUI) portion of EMail is generated by using Domain Dialogue software.

Processing Overview

The *EMail* function consists of a single executable file named **email** located in the **/etms5/email** directory. The **email** file is the result of binding the *email_m* program with the modules listed below. The *email_m* program is the initial entry point when **email** is executed. The processing performed by the *EMail* function is grouped into the following modules:

- *email_p* - This module contains general routines related for initialization and file handling.
- *email_new_p* - This module contains routines related to address processing and dialogue button selection processing.

11.1 The email_p Module

Purpose

The *email_p* module contains related general routines for initialization and file handling. Figure 11-2 shows the task flow of the *email_p* routine organization.

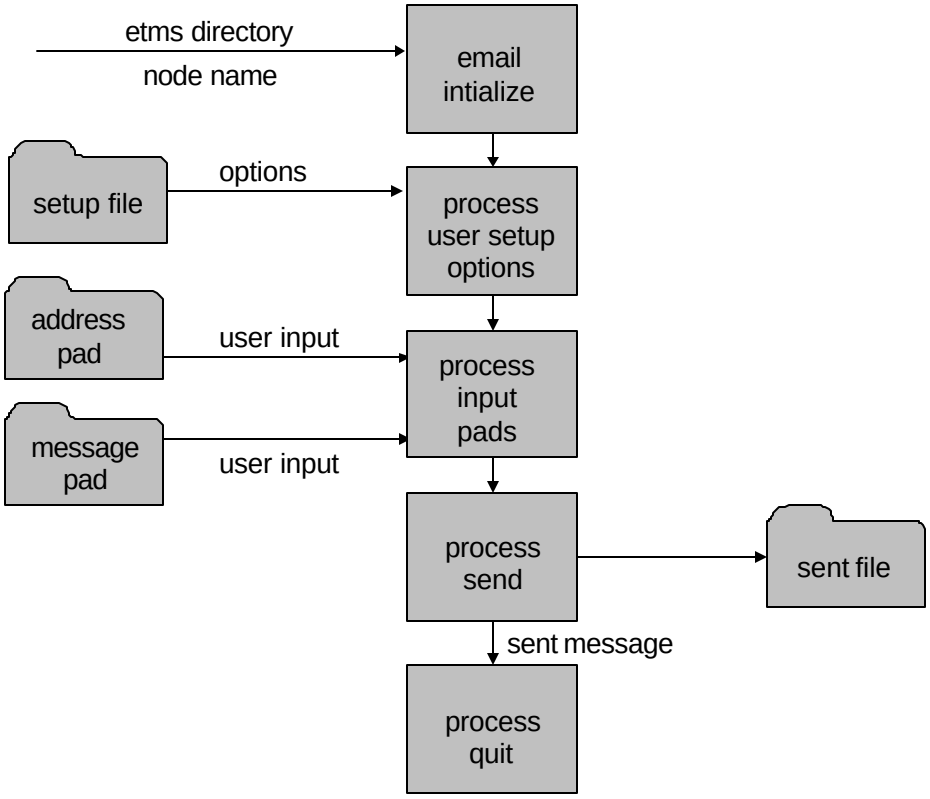

Figure 11-2. Routine Organization for the email_Module

11.1.1 The break_down_node_list Routine

Purpose

The *break_down_node_list* routine breaks down the list of nodes used for advisory number processing.

Input

node list string - long string (1,024 characters)

Output

node array - narray of node_record types

node count

Processing

The *break_down_node_list* routine performs the following tasks:

(1) Sets **node_count** to zero.

(2) If **str_len** is greater than zero, calls the *email_get_word* routine to get each node in the list and puts them into an array called nodes.

Error Conditions and Handling

None

Routine Interfaces

email_get_word

11.1.2 The build_global_paths Routine

Purpose

The *build_global_paths* routine adds the root level ETMS directory name to constant path names for *EMail* to locate files and directories.

Input

global variables *etms_dir* and *etms_dir_len*

Output

full pathnames for files required by EMail and the length of each file

Processing

The *build_global_paths* routine builds the following paths for the identified file:

/shared/data/log_files/user_id for the *user_id_file.*

/shared/data/address_groups for the *keyword_dir_name.*

/shared/data/rcvd_msgs/ for the *rcv_msg_path_name.*

/email/data/saved_msgs/ for the *save_msg_path_name.*

/email/data/sent_addrs/ for the *save_addrs_path_name*.

/email/data/sent_msgs/ for the *sent_msg_path_name.*

/email/data/std_msgs/ for the *std_msg_path_name.*

/email/data/address_files/address_mappings for the *alt_addr_file.*

/shared/adapt/print_script for the *print_script.*

/email/adapt/print_script for the *email_ print_script.*

/printers/prnec for the *default_print_script.*

/email/config/email_function_enable_table for the *func_enable_fname.*

/shared/data/log_files/adv_file for the *adv_fname.*

/email/data/address_files/nadin_addresses for the *nadin_addresses_file.*

/email/data/address_files/arnc_addresses for the *arnc_addresses_file.*

/email/data/address_files/etms.

/shared/fonts/etms_icon.1.

/email/data/tmp/report_list.

/email/adapt/groups_to_list.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.3 The check_nadin_arinc_data Routine

Purpose

The *check_nadin_arinc_data* routine checks the **nadin_arinc_data** file text strings for errors.

Input

nadin_arinc_data file global variable

Output

nadin_arinc_data record of type nadin_arinc_rec_t

Processing

The *check_nadin_arinc_data* routine

(1) Compares, using *strncpy*, against error validation key strings the **nadin_arinc_data** file text strings: NADIN source, ARINC source, phone number, and advisory prefix.

(2) The value is copied using *strncpy* if a match is found.

(3) A message might be displayed to suggest checking the **nadin_arninc_data** file for invalid data.

A message might be displayed to suggest checking the **nadin_arinc_data** file for invalid data.

Error Conditions and Handling

If invalid data is found, the following message is displayed, using *display_popup_text*:

Check the **/shared/config/nadin_arinc_data** file for invalid data.

Routine Interfaces

strncmp

strncpy

display_popup_text

11.1.4 The copy_list Routine

Purpose

The *copy_list* routine copies a file list.

Input

original file list, a pointer to *filedata_rec_t* type

Output

file list copy, a pointer to *filedata_rec_t* type

Processing

The *copy_list* routine performs the following tasks:

(1) Sets **source_pointer** to **orig_file_list**.

(2) Allocates memory for **target_ptr**.

(3) Sets **target_ptr** to NIL if **source_ptr** is NIL.

(4) If **target_ptr** is not NIL, copies the file list.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.5 The determine_addr_type Routine

Purpose

The *determine_addr_type* routine checks an address and sets a type for the address.

Input

node address - a string

node address length

Output

node address type - an enumerated type (aft, arn, apo, flow, undef)

Processing

The *determine_addr_type* routine searches the input address and sets a type for the address according to the following conditions:

- If the address contains a ".", and the "." is followed by at least three characters, the address is classified as an ETMS address.
- If the address is seven characters long, the address is classified as an ARINC address.
- If the address is eight characters long, the address is classified as a NADIN address.

- If the address is not ETMS, ARINC, or NADIN, it is classified as undefined and the program returns to the calling procedure.
-

Error Conditions and Handling

None

Routine Interfaces

None

11.1.6 The display_st_msg Routine

Purpose

The *display_st_msg* routine displays messages passed back from the *new_com_routines*.

Input

status message string

status message length

Output

standard error message output is displayed.

Processing

The *display_st_msg* routine checks the length of the string passed and displays messages passed back from the *new_com_routines* according to the following conditions:

- If the string is less than 1 character, the routine returns to the calling procedure.
- If the status message is **no_msg,** the routine returns to the calling procedure.
- If the status message is an **info**, *display_popup_text* is called to display the message and **st_msg** is set to **no_msg.**
- If the status message is an **error**, *display_popup_text* is called to display the error message and **st_msg** is set to **no_msg**.
- If the status message is **fatal**, *display_popup_text* is called to display the error message.
-

Error Conditions and Handling

None

Routine Interfaces

display_popup_text

11.1.7 The display_popup_text Routine

Purpose

The *display_popup_text* routine displays messages to the user in a popup form.

Input

popup text - univ of 256 characters

text length - an integer

task_id - dialog type of *dp_#task_id*

alert - boolean

Output

None

Processing

The *display_popup_text* routine performs the following tasks:

(1) If an error message was displayed, returns to the calling procedure.

(2) If the error message text length is greater than the maximum error message length, gets the last blank position in the error message and subtracts one character position to determine the position of the last character.

(3) Determines whether the message to be displayed is an alert message or a status message and calls *dp_$msg_set_value* to set the message text in the task. If the *task_id* equals *init_display_text_task*, *wait_for* is called to keep the message displayed for a specified amount of time.

(4) Displays the appropriate message to the user, calls *email-reset-activation-count* to reset the activation count, and gets the stored value in *popup_offset* to determine where the message is to be displayed.

Error Conditions and Handling

None

Routine Interfaces

email_reset_activation_count

wait_for

11.1.8 The email_addr_cancel Routine

Purpose

The *email_addr_cancel* routine clears the addresses that have been checked on the interface and pops down the SELECT ADDRESSES menu.

Input

address menu task *dp_s/task_id*

Output

reset address menus - dialog menu arrays

Processing

The *email_addr_cancel* routine performs the following tasks:

(1) Initializes the addresses array by calling *email_clear_address_array.*

(2) Checks each address type menu for selected addresses.

(3) Sets each selected address to the deselect value by calling *email_reset_menu* and pops down the SELECT ADDRESSES menu.

(4) Calls *email_display_menu* choices to display the previously selected addresses.

Error Conditions and Handling

None

Routine Interfaces

email_clear_address_array

email_reset_menu

email_display_menu_choices

11.1.9 The email_break_up_line Routine

Purpose

The *email_break_up_line* routine takes the line passed into the routine and breaks it up so the maximum line length is **max_chars_per_line** and writes those lines to the indicated data stream.

Input

line - long string

line length

stream

maximum number of characters allowed per line - integer

Output

broken up lines written to data stream

Processing

The *email_break_up_line* routine performs the following tasks:

(1) Checks the length of the line passed into the routine. If the line is longer than the maximum line length allowed, the line is broken up so the maximum line length is max_chars_per_line.

(2) Calls *ios_$put* to write the lines to the indicated data stream.

Error Conditions and Handling

None

Routine Interfaces

ios_$put system call

11.1.10 The email_check_on_quit Routine

Purpose

The *email_check_on_quit* routine checks whether the address pad or message pad has been changed since the last send.

Input

address pad - an edit pad

message pad - an edit pad

Output

confirmation popups for "Quit" and "message not saved or sent"

Processing

The *email_check_on_quit* routine performs the following tasks:

(1) Checks whether the "confirm on quit" setup option is selected. If it is not selected, *display_popup_text* is called to display "Shutting down EMail" and *EMail* will save and close any open pads by calling *goto_position*, *screen_position*, and *close_window*, and then *EMail* shuts down.

(2) Checks whether the message edit pad is empty by calling *file_length*. If it is not empty, the message will be checked to see if it is the same as the last message sent. If the message is not the last message sent, *email_reset_activation_count* is called to display a popup window asking for the quit to be confirmed.

Error Conditions and Handling

None

Routine Interfaces

close_window

display_popup_text

file_length

email_reset_activation_count

screen_position

goto_position

11.1.11 The email_clear_address_array Routine

Purpose

The *email_clear_address_array* routine clears the address array by setting the **enttype** to a deselect value (3).

Input

type name_array_rec address array

Output

deselect value

Processing

The *email_clear_address_array* routine sets all addresses in the address array to the deselect value of three.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.12 The email_clear_addrs Routine

Purpose

The *email_clear_addrs* routine clears the addresses that have been checked on the interface and pops down the SELECT ADDRESSES menu. If this routine is called from the main CLEAR button on the interface, the address pad is also cleared.

Input

address array

type name_array_rec

Output

cleared address array

Processing

The *email_clear_addrs* routine performs the following tasks:

(1) Calls *screen_location* to save the location of the cursor. Then it calls *email_clear_address_array* to initialize all menu types.

(2) All address types are checked for selected addresses. If an address is selected, the **enttype** field is set to the deselect value and *email_reset_menu* and *email_display_menu_choices* are called to clear selected choices. The menu is popped down.

(3) If the **task_id** is equal to **clear_addr_task**, the address pad is also cleared by calling *move_to_window* and *goto_position.* The *display_popup_text* routine is called to display "Processing" while the clear is executed. The *email_reset_activation_count* routine is called to pop down the "Processing" message.

Error Conditions and Handling

None

Routine Interfaces

screen_location

email_clear_address_array

email_reset_menu

email_display_menu_choices

display_popup_text

move_to_window

goto_position

email_reset_activation_count

11.1.13 The email_clear_addrs_pad Routine

Purpose

The *email_clear_addrs_pad* routine clears the address input pad area.

Input

address pad - edit pad

Output

cleared address pad - edit pad

Processing

The *email_clear_addrs_pad* routine performs the following tasks:

(1) Sets **position** to the **screen_location** and then calls *move_to_window* to position the cursor in the window.

(2) Calls *pad_$dm_cmd* to execute the Display Manager commands to clear the address pad.

(3) Calls *goto_ position* to return the cursor to its previous screen position.

Error Conditions and Handling

None

Routine Interfaces

screen_location

move_to_window

goto_position

pad_$dm_cmd

11.1.14 The email_close_up_shop Routine

Purpose

The *email_close_up_shop* routine prepares for the termination of *EMail* by deleting the input pads and closing the communication channel.

Input

open *EMail* pads - edit pads

Output

closed *EMail* pads - edit pads

Processing

(1) Displays the popup-text "Shutting down EMAIL".

(2) Calls *pad_$set_icon_font* to change the icon font for the specified window.

(3) Calls *pad_$set_icon_pos* to establish an icon position and character for the specified window.

(4) Calls *ios_$close* to close the address pad stream.

(5) Calls *ios_$close* to close the edit pad stream.

(6) Calls *wait_for* to wait 1 second.

(7) Calls *name_$delete_file* to delete the address pad.

(8) Calls *name_$delete_file* to delete the edit pad.

(9) Calls *email_reset_activation_count* to initialize the *init_popdown_task.*

(10) Calls *dp_$task_activate* to activate the *init_popdown_task*.

(11) Calls *dp_$terminate* to terminate *the dialog for EMail*.

(12) Calls *close_com* to close the communication channel.

Error Conditions and Handling

None

Routine Interfaces

*display_popup_tex*t

wait_for

email_reset_activation_count

11.1.15 The email_create_addr_pad Routine

Purpose

The *email_create_address_pad* routine creates the address pad at the top of the screen. This routine tries to open a window on **addr_pad_name**. If that file already exists, a new name is created and a window is opened.

Input

addr_pad_name

name_$long_pname

Output

address.pad

pad_$edit

Processing

(1) Initializes *edit_pad_name* and *edit_pad_path*.

(2) Gets the full pathname of the input pad and checks whether the path exists. If the path does not exist, *display_popup_text* is called to display "Error creating a message input area. EMail will exit."

(3) Calls *vfmt_$encode* to put the pathname in front of the file name.

(4) Sets append_num to 1.

(5) Attempts to delete the file.

(6) Creates the input pad file as long as the file does not already exist. If the file does exist, an attempt is made to delete the file. If the file cannot be deleted, a number is appended to the file name and append_num is incremented.

(7) Closes the file and opens a window on it.

(8) Disables the SAVE, EXIT, ABORT, ^Y, ^N, and ^M function keys on the edit pad.

(9) Calls *move_to_window* to move the cursor to the window.

(10) Sets the input line length to 68 characters.

Error Conditions and Handling

Calls *display_popup_text* to display the following error message if an error is encountered during processing:

Error creating an address input area. EMail will exit!

Routine Interfaces

display_popup_text

move_to_window

11.1.16 The email_create_edit_pad Routine

Purpose

The *email_create_edit_pad* routine creates the input pad at the bottom of the screen. This routine tries to open a window on *edit_pad_name*. If that file already exists, a new name is created and a window is opened.

Input

edit_pad_name

name_$long_pname

Output

edit pad

pad_$edit

Processing

The *email_create_edit_pad* routine performs the following tasks:

(1) Initializes *edit_pad_name* and *edit_pad_path*.

(2) Gets the full pathname of the input pad and checks whether the path exists. If the path does not exist, *display_popup_text* is called to display "Error creating a message input area. EMAIL will exit."

(3) Calls *vfmt_$encode* to put the pathname in front of the file name.

(4) Sets append_num to 1.

(5) Attempts to delete the file.

(6) Creates the input pad file as long as the file does not already exist. If the file does exist, an attempt is made to delete the file. If the file cannot be deleted, a number is appended to the file name and append_num is incremented.

(7) Closes the file and opens a window on it.

(8) Disables the SAVE, EXIT, ABORT, ^Y, ^N, and ^M function keys on the edit pad.

(9) Calls *move_to_window* to move the cursor to the window.

(10) Sets the input line length to 68 characters.

Error Conditions and Handling

Calls *display_popup_text* to display an error message if an error is encountered during processing. The error message reads:

Error creating a message input area. EMail will exit!

Routine Interfaces

display_popup_text

move_to_window

11.1.17 The email_create_ui_addr_header_pad Routine

Purpose

The *email_create_ui_addr_header_pad* routine creates the address header pad above the address pad.

Input

transcript pad text

Output

header pad - transcript_$pad

address

Processing

The *email_create_ui_addr_header_pad* routine performs the following tasks:

(1) Calls *pad_$create* to create a new transcript pad at the bottom of the existing window that is not reachable by the cursor.

(2) Calls *ios_$put* to display "ENTER ADDRESSES BELOW" on the message header pad.

(3) Calls *pad_$create* to create a new transcript pad at the bottom of the existing window, not reachable by the cursor.

(4) Calls *pad_$create_frame* to create a frame on the advisory header pad.

(5) Calls *ios_$put* to display "GENERAL MESSAGE" on the advisory header pad.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.18 The email_create_ui_msg_header_pad Routine

Purpose

The *email_create_ui_msg_header_pad* routine creates the message header pad above the message input pad.

Input

transcript pad text

Output

message header pad - pad_$transcript

Processing

The *email_create_ui_msg_header_pad* routine performs the following tasks:

(1) Calls *pad_$create* to create a new transcript pad at the bottom of the existing window that is not reachable by the cursor.

(2) Calls *ios_$put* to display "ENTER MESSAGE TEXT BELOW" on the message header pad.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.19 The email_delete_addr_pad Routine

Purpose

The *email_delete_addr_pad* routine closes and deletes the input pad file.

Input

open address pad stream - *ios_$id_t*

Output

closed open address pad stream - *ios_$id_t*

deleted address file name

Processing

The *email_delete_addr_pad* routine performs the following tasks:

(1) Calls *ios_$close* to close the edit pad stream.

(2) Waits for two seconds.

(3) Calls *name_$delete_file* to delete the edit input pad file.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.20 The email_delete_edit_pad Routine

Purpose

The *email_delete_edit_pad* routine closes and deletes the input pad file.

Input

open edit pad stream - *ios_$id_t*

Output

closed edit pad stream - *ios_$id_t*

deleted edit pad filename

Processing

(1) Calls *ios_$close* to close the edit pad stream.

(2) Waits for two seconds.

(3) .Calls *name_$delete_file* to delete the edit input pad file.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.21 The email_extract_body_of_message Routine

Purpose

The *email_extract_body_of_message* routine takes the message text from the input file and writes it to the file specified as **out_file**.

Input

in_file - string of 80 characters

in_file length

save_flag - boolean

over_write - boolean

Output

out_file - string of 80 characters

out_file length

Processing

The *email_extract_body_of_message* routine performs the following tasks:

(1) Attempts to open the input message file. If an error is encountered on the open, *display_popup_text* is called to display an error message.

(2) The *ios_$create* system call is called to create the message file. If the input message is to overwrite an existing message, the *ios_$recreate_mode* option is

used; otherwise, the *ios_$no_pre_exist_mode* option is used. If the *ios_$no_pre_exist_mode* option returns *ios_$already_exists* status, a popup window is displayed that asks if the new message should be saved over the old message. If there are any other errors on the create, *display_popup_text* is called to display an error message, the streams are closed, and the routine exits.

(3) If the message is being extracted to be saved, **add_phonenum** is set to FALSE.

(4) Calls *email_find_first_non_blank_line* to find the first line of the input message.

(5) If the message is being extracted for sending and it is an advisory message, *vfmt_$encode* is called to write ADVZY into the message string.

(6) Skips the phone number if it is included in the message and calls *ios_$put* to write the message to the output stream.

(7) Checks if the line is greater than **max_msg_len**. If the line is longer than the maximum line length, *email_break_up_line* is called to break up the message line and *ios_$put* is called to write the message to the output stream.

(8) Checks if the phone number is to be appended to the message and calls *ios_$put* to append the phone number to the output message stream, and **add_phonenum** is set to FALSE.

(9) If **add_phonenum** is TRUE, *vfmt_$encode* is called to add the phone number to the output message stream before sending the message.

(10) Closes the input and output streams.

Error Conditions and Handling

Calls *display_popup_text* to display an error message and exits the routine if an error is encountered during processing. The error message reads:

Error: Could not open input message file."

Routine Interfaces

display_popup_text

email_reset_activation_count

strncmp

email_break_up_line

email_find_first_non_blank_line

11.1.22 The email_file_empty Function

Purpose

The *email_file_empty* function checks to see whether a file has characters in it.

Input

file name - a string of characters

file length

Output

None

Processing

The *email_file_empty* function performs the following tasks:

(1) Calls *ios_$open* to open the indicated file.

(2) Calls *email_find_first_non_blank_line* to determine if there are any characters in the file.

(3) Calls *ios_$close* to close the file.

Error Conditions and Handling

None

Routine Interfaces

find_first_non_blank_line

11.1.23 The email_find_first_non_blank_line Routine

Purpose

The *email_find_first_non_blank_line* routine returns the first non blank line from the stream ID passed in.

Input

file stream - *ios_$id_t*

Output

buffer - a string of 1024 characters

eof flag – Boolean

Processing

The *email_find_first_non_blank_line* routine performs the following tasks:

(1) Calls *ios_$get* to read each line from the indicated stream and determines if it is blank or non-blank until the first non-blank line is found. If an error is encountered on the get, the routine exits.

(2) Calls *get_word_from_buffer* to get the first non-blank line from the buffer.

Error Conditions and Handling

None

Routine Interfaces

get_word_from_buffer

11.1.24 The email_generate_advisory_number Routine

Purpose

The *email_generate_advisory_number* routine maintains the advisory number sequence and places the number in the current message file to be sent.

Input

number of advisory files

current advisory number

Output

updated advisory number

Processing

The *email_generate_advisory_number* routine performs the following tasks:

(1) Checks the network for any advisory files by calling *update_adv_number*.

(2) If there are advisory files, the popup text "Updating advisory numbers around the network" is displayed, and the advisory number is updated.

(3) Calls *email_reset_activation_count* to reset the activation count of the *init_popdown_task*.

(4) Calls *dp_$task_activate* to start the *init_popdown_task.*

(5) If there are no advisory files, the popup text "Warning! - Unable to update advisory numbers around the network" is displayed and the advisory number is incremented.

Error Conditions and Handling

If the number of advisory files is zero, *display_pop_text* is called to display the following warning:

Unable to update advisory numbers around the network.

Routine Interfaces

display_popup_text

update_adv_number

email_reset_activation_count

11.1.25 The email_get_args Routine

Purpose

The *email_get_args* routine gets the node name and stores the value in global variables, sets global path names, and determines the *EMail* functions to enable.

Input

node name

Output

global pathnames used by EMail

Processing

(1) Calls *name_$get_path_lc* to get the current node name. If bad status, a **node_id_len** of 0 or a null **node_id** is returned, and *display_popup_text* is called to display an error message.

(2) Calls *get_etms_path* to get the ETMS directory length. If **etms_dir_len** is less than 1, *display_popup_text* is called to display an error message, dir_struct_ok is set to FALSE and the routine exits. If **etms_dir_len** is not less than 1, it gets the full pathname of the ETMS directory and checks the status. If a bad status is returned, *display_popup_text* is called to display an error message and the routine exits.

(3) Calls *build_global_paths* to build the global path names.

(4) Sets the **address_file_names** array indices.

(5) Calls *name_$get_path*_lc to get the full path name of the function enable table. If a bad status is returned, the routine exits.

(6) Calls *email_parce_func_arg*, which reads the function enable table to determine the *EMail* functions to enable.

Error Conditions and Handling

Calls *display_popup_text* to display an error message if the routine could not set the node name or the **etms** directory. The following error messages are displayed:

Could not set the node name search node set to All

Could not access required ETMS directory structure

Routine Interfaces

display_popup_text

get_etms_path

build_global_paths

email_parse_func_arg

11.1.26 The email_get_user_id Routine

Purpose

The *email_get_user_id* routine reads the data in the **user_id** file to get the **user_id**.

Input

user identification filename (*user_id*)

Output

user identification read from the file

Processing

(1) Initializes variables **user_id**, **i**, and **buf**.

(2) Calls *ios_$open* with the *ios_$read_intend_write* option to open the **user_id** file with read access. If the open is not successful, it will try again (up to 25 times) until it is successful.

(3) If the open is successful, the *ios_$get* procedure is called to read the data in the **user_id** file.

(4) If the **user_id** contains more than 4 bytes the popup text "The USER ID has been truncated to 3 chars" is displayed. The popup text "Initializing" is then displayed.

(5) The index is set to 1 to read the first character in the **user_id** file.

(6) As long as the indexed character is not a carriage return or a blank, and the index is less than 4, each character in the **user_id** file is checked. If the character is alphabetic it is placed in **user_id**.

(7) If the indexed character is a carriage return or a blank, or if the index is greater than 4, the popup text "The USER ID is undefined" is displayed. The popup text "Initializing" is then displayed. The **user_id** is set to **uid** and the **user_id_len** is set to **3**.

(8) If the status is not 0, the popup text "The USER ID is undefined" is displayed. The popup text "Initializing" is then displayed. The **user_id** is set to **uid** and the **user_id_len** is set to **3**.

(9) *ios_$close* is called to close the data stream.

(10) Gets the **user_id** from the **user_id** file and converts it to lower case by calling the *tolower* routine.

Error Conditions and Handling

Calls *display_popup_text* to display an error message for the following conditions:

(1) if the **user_id** is greater than three characters, the following message is displayed:

The USER ID has been truncated to 3 chars.

(2) If the **user_id** is not found or is not in the file, the following message is displayed:

The USER ID is undefined.

Routine Interfaces

display_popup_text

tolower

11.1.27 The email_get_word Routine

Purpose

The *email_get_word* routine returns a word from a buffer starting from where the pointer is pointing. If the end of the buffer is reached, the returned length is zero.

Input

text string

length of text string

pointer to search start position

maximum word size

Output

pointer to next search position

word - parsed from text string

length of word

Processing

The *email_get_word* routine performs the following tasks:

(1) Increments the **pointer** while it is less than **buf_len** and it is pointing to a line_feed, a null character, or a new line character.

(2) Gets the word that the **pointer** is pointing to while the **pointer** is less than or equal to **buf_len**, and is not pointing to a **line_feed**, null character, or new line character.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.28 The email_init Routine

Purpose

The *email_init* routine is a supervisory routine that calls other routines to perform initialization for *EMail* processing.

Input

enable valuesread from *email_function_erable_table*

user identification

adaptation values read from the *nadin_arinc_data* file

setup values read from the **setup** file

Output

All adaptation, configuration, and setup values set for the current EMail session.

Processing

The *email_init* routine performs the following tasks:

(1) Sets the icon font.

(2) Calls *email_get_args* to get the node name, set the global path names, and determine the *EMail* functions to enable.

(3) Checks the directory structure. If **dir_struct_ok** is FALSE, the routine exits.

(4) Calls *email_get_user_id* to get the **user_id**.

(5) Calls *read_lcnode_list* to construct a list of nodes on the network and the time offsets between the nodes.

(6) Calls *create_file_list* to create the file list.

(7) Calls *set_adaptation_values* to set the adaptation values.

(8) Calls *init_user_message_record* to initialize the user message record.

(9) Calls *email_read_alt_addr* to read the alternate address list.

(10) Calls *email_set_address_choices* to set up the array of available addresses to be loaded into the address menu in the user interface.

(11) Calls *email_set_keyword_choices* to obtain a list of all keywords for use by *EMail* and displays this list.

(12) Calls *email_create_ui_addr_header_pad* to create the header on the address pad.

(13) Calls *email_setup* to set up *EMail*.

Error Conditions and Handling

The routine exits if a bad directory structure is found. The error message is reported by calling *email_get_args*.

Routine Interfaces

email_get_args

read_lcnode_list

create_file_list

set_adaptation_values

init_user_msg_record

email_get_user_id

email_read_alt_addr

email_set_address_choices

email_set_keyword_choices

email_create_ui_addr_header_pad

email_setup

display_popup_text

11.1.29 The email_make_contains_file_list Routine

The *email_make_contains_file_list* routine creates a contains-file list.

Input

original-file list

contains string

contains string length

Output

original file list

Processing

The *email_make_contains_file_list* routine performs the following tasks:

(1) Calls *strncmp* to determine whether files exist in the file list.

(2) Creates a temporary file list by calling *contains* with the *contains* string to filter the list to display in the user interface.

(3) Returns if no files are found or a SEARCH ERROR is encountered.

Error Conditions and Handling

None

Routine Interfaces

strncmp

contains

str_len

11.1.30 The email_make_display_file_list Routine

The *email_make_display_file_list* routine creates a display-file list.

Input

original-file list

Output

display-file list

Processing

(1) Calls *copy_list* to copy the original-file list to a display-file list.

(2) Creates a temporary file list to display in the user interface.

Error Conditions and Handling

None

Routine Interfaces

copy_list

11.1.31 The email_make_icon Routine

Purpose

The *email_make_icon* routine pops the user interface into an icon.

Input

icon character

Output

EMail interface in icon form

Processing

The *email_make_icon* uses *pad_$make_icon* to change the existing full-size window into an icon.

Error Conditions and Handling

None

Routine Interfaces

pad_$make_icon

11.1.32 The email_make_save_name Routine

The *email_make_save_name* routine creates the name under which the current message is to be saved for sending. The name is derived from the user id, message type, date, time, and sequence number.

Input

filename

size of filename

Output

filename

size of filename

Processing

The *email_make_save_name* routine performs the following tasks:

(1) Determines whether the *EMail* message type is **advzy**. If it is, the message is assigned an advisory number and is classified as *ADVY*. If the message type is not **advzy** then it is classified as *GENR*.

(2) Calls *cal_$decode_local_time* to get the local date and time.

(3) Checks the sequence number. If the sequence number is greater than **99** it is reset to **0**.

(4) Calls *vfmt_$encode* to build the pathname for the file.

(5) Increments the sequence number.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.33 The email_parse_func_arg Routine

Purpose

The *email_parse_func_args* routine parses a string containing functions to be enabled in *EMail* and constructs a global set of enabled functions.

Input

function argument string

function argument length

Output

set of enable functions

Processing

The *email_parse_func_args* routine performs the following tasks:

(1) Initializes enabled_functions and temp_func_arg.

(2) Calls *vfmt_$encode* to convert the string to upper case, replacing non-blank delimiters with spaces.

(3) Calls *strncmp* to determine whether the functional argument string is ARINC, NADIN, or ADZY and enables the functions according to the argument string.

Error Conditions and Handling

None

Routine Interfaces

strncmp

11.1.34 The email_print_file Routine

Purpose

The *email_print_file* invokes a script to print the message pad.

Input

file name

file name length

node name

node name length

Output

print error

print error length

Processing

(1) Gets the path of the print script to be invoked and sets **print_command** to the appropriate print script. If the path is not found, sets **print_command** to a default print script.

(2) Sets up the arguments for invocation of the print script.

(3) Calls *pgm$invoke* to invoke the indicated print script. Writes an error message to **print_error** if the print script cannot be invoked.

.

Error Conditions and Handling

Calls *display_popup_text* to display the following error message if the print script could not be found:

Could not find: *print_script*.

Routine Interfaces

None

11.1.35 The email_print_file_at_node Routine

Purpose

The *email_print_file_at_node* routine invokes a script to copy a file to another node's print queue.

Input

file name

file name length

node name

node name length

Output

print error string

print error string length

Processing

The *email_print_file_at_node* routine performs the following tasks:

(1) Writes the file to be copied to **extra_copy_print_script** and gets the path to the file. If the path does not exist, writes an error message to **print_script**.

(2) Sets up the arguments for the invocation of **extra_copy_print_script**.

(3) Calls *pgm_$invoke* to copy the file to the indicated node's print queue. If the copy is not successful, an error message is written to **print_error**.

Error Conditions and Handling

If the print script could not be found, the following error message is passed back to the calling routine:

Could not find: *print_script*.

Routine Interfaces

None

11.1.36 The email_process_address_pad Routine

Purpose

The *email_process_address_pad* routine processes the addresses in the address input pad.

Input

ETMS, NADIN, and ARINC addresses

Output

error string

length of error string

Processing

The *email_process_address_pad* routine performs the following tasks:

(1) Opens the **address_pad_name** file. If an error is encountered on the open, writes an error message to **error_str**, closes the stream to **address_pad_name**, and exits.

(2) Calls *email_get_word* to parse addresses.

(3) Calls *email_conv_ifcn_to_vi* to convert the user interface address to its corresponding IFCN address and calls *determine_addr_type* to determine the address type. If the address type is undefined or disabled, an error message is written to **err_str** and the routine proceeds to the next address.

(4) Checks whether the chosen address is in the address array. If the address is in the address array, it is selected. If the address is not present in the address array, the routine adds the chosen address. If no addresses are selected, an error message is written to **error_str**.

Error Conditions and Handling

(1) Returns an error if the address pad cannot be opened.

(2) Returns an error if no addresses were found.

Routine Interfaces

email_get_word

determine_addr_type

email_conv_ifcn_to_ui

11.1.37 The email_process_apply_address Routine

Purpose

The *email_process_apply_address* routine processes the selection of addresses, groups the addresses, and places the address groups in the address pad area.

Input

addresses and groups selected by the user

Output

addresses displayed in the address input pad

Processing

The *email_process_apply_address* routine performs the following tasks:

(1) If no error message was displayed, the variable *popup_offset* is set to 1 for the address status popup.

(2) Sets the old group name and length to the value of the current group name and length and sets the message in the *main_group_display_text_task* and *addr_group_display_text_task*.

(3) The *email_reset_menu* is called to reset the *address_keyword_menu_task* and the *keyword_menu_task*.

(4) Determines the address type and sets the address to the "selected" value.

(5) Writes the address selections to the input pad. The *email_conv_ifcn_to_ui* is called to take the passed address, looks for a matching address in a list of alternate address names, and replaces the address with a corresponding name to be displayed in the user interface.

(6) *Move_to_window* is called to move the cursor to the place specified in the address pad stream in the *EMail* display.

(7) Sets the correct menu task to correspond to the address type, APOLLO, ARINC, or NADIN.

(8) Calls *email_reset_menu* to reset the new address keyword menu task and the keyword menu task and then calls *goto_position* to move the cursor to its previous position.

Error Conditions and Handling

None

Routine Interfaces

email_reset_menu

email_conv_ifcn_to_ui

move_to_window

goto_position

11.1.38 The email_process_check_print Routine

Purpose

The *email_process_check_print* routine displays the confirm_on_print popup if the option is set.

Input

global variable *confirm_on_print*

Output

processed print command

Processing

The *email_process_check_print* routine checks whether the **print_confirm** option is set. If it is set, the **confirm_on _print** popup is displayed by calling *email_reset_activation_ count* before the file is printed. If it is not, the *email_process_print* routine is called.

Error Conditions and Handling

None

Routine Interfaces

email_reset_activation_count

email_process_print

11.1.39 The email_process_check_send Routine

Purpose

The *email_process_check_send* routine displays the confirm_on_send popup if the option is set.

Input

global variable *confirm_on_send*

Output

processed send command

Processing

The *email_process_check_send* routine checks whether the **send_confirm** option is set. If it is set, the **confirm _on _send** popup will be displayed by calling *email_reset_activation_count* before the message is sent. If it is not set, the *email_process_send* routine is called.

Error Conditions and Handling

None

Routine Interfaces

email_reset_activation_count

email_process_send

11.1.40 The email_process_file_type Routine

Purpose

The *email_process_file_type* routine processes the user's file type selection to display in the IMPORT file menu.

Input

file type, reports, standard messages, saved messages, and sent messages

Output

directory listing of selected file type

Processing

The *email_process_file_type* routine performs the following tasks:

(1) Gets the file type selection and sets the path to that selection.

(2) Calls *filename_search* to search the selected type directory and display the files in the IMPORT file menu. The SEARCH popup is displayed during the search. If an error is encountered during the search, *display_popup_text* is called to display an error message.

(3) If the number of reports matching the search criteria exceeds the program limit, calls *display_popup_text* to display an error message.

Error Conditions and Handling

Calls *display_popup_text* to display the following error message if the number of reports exceeds the program limit of 1360:

Warning-number of reports matching search criteria exceeds 1360 program limit.

Routine Interfaces

filename_search

display_popup_text

11.1.41 The email_process_options Routine

Purpose

The *email_process_options* routine processes the user-selected setup options.

Input

setup option menu choices

Output

new setup selections written to the *setup* file

Processing

The *email_process_options* routine performs the following tasks:

(1) If the CANCEL button is selected, defaults to the previously selected options.

(2) If the OK button is selected, calls *strncmp* to check to see if any of the options have changed and exits if there were no changes. If changes were made, creates a new setup options file, **/etms/email/config/setup.uid**, calls *vfmt_$ws* to write the new option values to the file, and sets the new option value.

(3) Pops down the window.

Error Conditions and Handling

None

Routine Interfaces

strncmp

11.1.42 The email_process_reset Routine

Purpose

The *email_process_reset* routine clears the message text area.

Input

global task identification

Output

cleared message input pad

Processing

The *email_process_reset* routine performs the following tasks:

(1) Calls *screen_location* to set position to the screen_location.

(2) Calls *display_popup_text* to display a processing popup.

(3) Moves the cursor to the indicated window and calls *pad_$dm_cmd* to execute the Display Manager commands to clear the message pad. Then calls *email_reset_activat_count* to pop down to processing message.

(4) Calls *move_to_window* to return the cursor to its previous screen position.

Error Conditions and Handling

None

Routine Interfaces

screen_location

display_popup_text

move_to_window

email_reset_activation_count

goto_position

11.1.43 The email_process_save Routine

Purpose

The *email_process_save* routine saves the current contents of the input pad into a file.

Input

input edit pad

Output

contents of the input edit pad saved to a file

Processing

The *email_process_save* routine performs the following tasks:

(1) Calls *screen_location* to get and store the popup screen location.

(2) Gets the entered name, converts it to lower case ASCII characters and stores it in the file's pathname. If no name was entered, sets the pathname to "_".

(3) Searches for a "/" in the pathname. If the pathname does not contain a "/" the invalid pathname flag is set to TRUE, and *display_popup_text* is called to display an error message that the path/file name is invalid. If the pathname does contain a "/", determines the length of the pathname, and *name_$get_path_lc* is called to store the pathname.

(4) If the pathname does contain a "/", determines the length of the pathname, and *name_$get_path_lc* is called to store the pathname.

(5) The *move_to_window* routine is called to move the cursor to the specified place in the *EMail* window.

(6) The *email_file_empty* routine is called to determine whether the message is not empty. *EMail_extract_body_of_message* is called to take the message text from the input file and writes it to the specified output file. *Goto_position* is called to put the cursor at the previous position, and the activation count is reset.

(7) If no message is found, *display_popup_text* is called to display an error message that there was no message to save, *email_reset_activation_count* is called to reset the activation count, and the screen position is saved.

Error Conditions and Handling

The display_popup_text routine is called to display an error message to the effect that the path/file name is invalid or to display an error message specifying that there was no message to save.

Routine Interfaces

email_reset_activation_count

display_popup_text

move_to_window

email_extract_body_of_message

screen_location

goto_position

email_file_empty

11.1.44 The email_process_send Routine

Purpose

The *email_process_send* routine checks and sends the messages entered in the message input area.

Input

addresses and message text for sending

Output

sent message file

Processing

The *email_process_send* routine performs the following tasks:

(1) Determines whether *RTR* is active and, if it is, the address array is cleared.

(2) The *move_to_window* routine is called to move the cursor to the specified place in the *EMail* display and the procedure pauses for two seconds.

(3) The *email_process_address_pad* is called to process the address(es) in the address pad. If an error message is to be displayed, *display_popup_text* is called to display the error message. *Goto_position* is called to return the cursor to its previously saved screen location, and returns to the calling procedure.

(4) If no error text is to be displayed, *move_to_window* is called to move the cursor to the specified place in the *EMail* display and pauses for two seconds.

(5) If the file is empty, *display_popup_text* is called to display a message that there was no message to send, saves the screen location, cursor is returned to its previously saved screen location, and returns to the calling procedure.

(6) If the file contains an advisory message, the advisory header frame is cleared, and *email_generate_advisory_number* is called to maintain the advisory number sequence and place the number in the current message file, which is to be sent.

(7) The *EMail_make_save_name* routine is called to create a name for the current message to be saved in for sending the file. A path is built for the message to be sent and the path and addresses are input into the save_addrs_name file.

(8) Constructs a filename and then saves the last message sent. The routine also calls the *email_save_addrs* to save the last sent message address pad for the current user. If an error occurs, *display_popup_text* is called to display an error message to the effect that the last filename of address could not be saved.

(9) If the message is an advisory message, the body of the message is extracted by calling *email_extract_body_of_message* and setting the message length for checking upon exit of the procedure.

(10) The *EMail_group_and_send* routine sends the message and if auto print is on, *email_print_file* is called to print the messages as they are sent. If an error occurs during print processing, *display_popup_text* is called to display an error message that a problem was encountered during spooling of the message. If no errors occur, *display_popup_text* is called to display a message that the current message is spooling.

(11) If the extra copy feature is enabled, *display_popup_text* is called to display a message on the status of the printing of the extra copy and *email_print_file_at_node* is called to spool the extra copy of the message at the specified node.

Error Conditions and Handling

The *display_popup_text* routine is called to display an error message. Two possible errors can occur, one if a file could not be created for sending and another if the communications channel is down.

Routine Interfaces

move_to_window

email_process_address_pad

display_popup_text

goto_position

email_generate_advisory_number

email_make_save_name

email_save_addrs

email_extract_body_of_message

email_group_and_send

email_print_file_at_node

email_file_empty

email_print_file

email_reset_activation_count

11.1.45 The email_process_type Routine

Purpose

The *email_process_type* routine processes the selection of the message type.

Input

message type - general, advisory

Output

message pad header

Processing

(1) Checks the selected message type.

(2) If the message type is **advzy**, gets the advisory number and puts the advisory in the message pad header field.

(3) If the message type is genr, GENERAL MESSAGE is put in the message pad header field.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.46 The email_quit_confirm Routine

Purpose

The *email_quit_confirm* routine displays a popup to notify the user that the **quit** option has been selected.

Input

quit confirm flag

Output

quit confirm popup

quit confirm save/sent message popup

Processing

The *email_quit_confirm* routine performs the following tasks:

(1) Checks the **quit_confirm** option. If the **quit_confirm** option is not selected, the routine checks the **quit_confirm_save** option. If **quit_confirm_save** is not selected, the routine returns to the calling procedure. If **quit_confirm_save** is TRUE, the routine calls *email_check_on_quit*.

(3) If **quit_confirm** is TRUE, a popup is displayed notifying the user that the **quit** option was selected.

Error Conditions and Handling

None

Routine Interfaces

email_check_on_quit

11.1.47 The email_read_alt_addr Routine

Purpose

The *email_read_alt_addr* routine constructs a global list of selected addresses and an alternate name for each address to be presented in the user interface.

Input

alternate addresses filename - /email/data/address_files/mappings

Output

address list of addresses and alternate addresses

Processing

The *email_read_alt_addr* routine performs the following tasks:

(1) Attempts to open a data stream to **alt_addr_file**. If the **alt_addr_file** does not exist, the routine exits.

(2) Calls *email_find_first_non_blank_line* to find the first line in the **alt_addr_file**.

(3) For each address in the address file, calls *get_word_from_buffer* once to get the address and again to get the alternate name for that address.

(4) Calls *ios_$close* to close the stream.

Error Conditions and Handling

The routine exits if the **alt_addr_file** cannot be opened.

Routine Interfaces

get_word_from_buffer

email_find_first_non_blank_line

11.1.48 The email_remove_duplicate_menu_choices Routine

Purpose

The *email_remove_duplicate_menu_choices* routine removes duplicate names.

Input

choice_array - array of menu choices

count - number of choices

Output

choice - array of menu choices with duplicates removed.

adjusted count

Processing

The *email_remove_duplicate_menu_choices* routine compares, using the *strncmp* routine, the names in the name array, and removes any duplicate names.

Error Conditions and Handling

None

Routine Interfaces

strncmp

11.1.49 The email_reset_activation_count Routine

Purpose

The *email_reset_activation_count* routine sets the activation count of a task back to **0**.

Input

task ID - dialog task identification

Output

activation count

Processing

The *email_reset_activation_count* routine performs the following tasks:

(1) Checks the activation count of the indicated task.

(2) If the activation count of the task is greater than **0**, *dp_$task_deactivate* is called to decrement the activation count until the activation count is **0**.

(3) If the activation count of the task is less than **0**, *dp_$task_activate* is called to increment the activation count until the activation count is **0**.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.50 The email_reset_groups Routine

Purpose

The *email_reset_groups* routine resets the EMail groups.

Input

task_id - dialog task identification

Output

all groups display in the groups menus

Processing

The *email_reset_groups* routine performs the following tasks:

(1) Checks the task ID and sets **task_name**.

(2) Calls *email_select_all_groups* to set the chosen task to display all groups.

Error Conditions and Handling

None

Routine Interfaces

email_select_all_groups

11.1.51 The email_save_addrs Routine

Purpose

The *email_save_addrs* routine takes the address text from the input file and writes it to the file specified as **out_file**.

Input

in_file - the address input pad name

in_file length - the length of the input pad name

Output

out_file - filename with saved addresses

out_file length - the length of the filename

Processing

The *email_save_addrs* routine performs the following tasks:

(1) Attempts to open the input address file. If an error is encountered on the open, *display_popup_text* is called to display an error message.

(2) Calls *ios_$create* to create the address file. If errors are encountered on the create, *display_popup_text* is called to display an error message. Calls *ios_$create* then exits the routine.

(3) Calls *email_find_first_non_blank_line* to find the first line of the address file.

(4) Calls *ios_$put* to write the address to the output stream.

(5) Closes the input and output streams.

Error Conditions and Handling

Calls *display_popup_text* to display an error message if the routine is unable to open the address file and exits the routine if an error is encountered during processing.

Routine Interfaces

display_popup_text

email-find-first-non-blank-line

11.1.52 The email_save_over Routine

Purpose

The *email_save_over* routine saves over a file if it already exists.

Input

save message filename

message text from the input pad

Output

message text saved in the specified filename

Processing

The *email_save_over* routine performs the following tasks:

(1) Gets the message to be saved by calling the *email_extract_body_of_massage* routine.

(2) If there are no errors displayed, a popup window is displayed by calling *email_reset_activation_count* asking whether the current file should be saved over the existing file. If *yes* is chosen, the file is saved over the existing file.

Error Conditions and Handling

None

Routine Interfaces

email_extract_body_of_message

email_reset_activation_count

11.1.53 The email_save_over_no Routine

Purpose

The *email_save_over_no* routine resets the save-over file to blank.

Input

over_write_file - filename

over_write_filelen - length of the filename

Output

initialized filename and length of zero

Processing

The *email_save_over_no* routine simply resets the **over_write_file** to blank.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.54 The email_save_cancel Routine

Purpose

The *email_save_cancel* routine sets the popup offset to zero

Input

popup_offset

Output

popup_offset set to zero

Processing

The *email_save_cancel* routine sets the global variable **popup_offset** to zero.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.55 The email_select_all_groups Routine

Purpose

The *email_select_all_groups* routine displays all the address group names in the group popup menus.

Input

popup_offset

address group menu array

Output

selected address groups

Processing

The *email_select_all_groups* routine performs the following tasks:

(1) Determines which popup window is displayed and sets the popup offset.

(2) If the popup window displayed is the GROUP menu or the GROUP menu within the SELECT ADDRESSES menu, any previous selections are deselected, by calling *email_set_keyword_choices,* and the new selection is displayed.

Error Conditions and Handling

None

Routine Interfaces

email_set_keyword_choices

11.1.56 The email_set_address_choices Routine

Purpose

The *email_set_address_choices* sets up an array of available addresses to be loaded into the address menu in the user interface.

Input

popup offset

addresses menu array

Output

loaded menu array

Processing

The *email_set_address_choices* performs the following tasks:

(1) Initializes the number of menu choices to zero in case of premature exit.

(2) Calls *email_clear_address_array* to clear the address array.

(3) Calls *strncmp* to check for disabled address types and prevents address loading for those address types.

(4) Opens the address file for the indicated address type. If there is an error on the open, **sets num_disp_addresses** to **0** and proceeds to the next address type.

(5) Calls **get_word_from_buffer** to get each address from the file and loads the addresses into the **address_group_array** until the **max_menu_entries** limit is reached.

(6) Calls *email_conv_ifcn_to_ui* to replace the IFCN address name with a corresponding alternate address name, if one exists.

(7) Calls *email_sort_menu_choices* to sort the addresses and calls *email_remove_duplicate_menu_choices* to get rid of any duplicate address names.

(8) Sets the **address_menu_task** and calls *email_display_menu_choices* to load the addresses into the address menu.

Error Conditions and Handling

If an error occurs while opening an address file, the number of addresses for that type is set to zero.

Routine Interfaces

email_clear_address_array

get_word_from_buffer

strncmp

email_conv_ifcn_to_ui

email_sort_menu_choices

email_remove_duplicate_menu_choices

email_display_menu_choices

11.1.57 The email_set_keyword_choices Routine

Purpose

The *email_set_keyword_choices* routine obtains a list of all keywords for use by *EMail* and displays this list.

Input

list for address groups used by *Email* **groups_to_list** file

Output

keyword list array

Processing

The *email_set_keyword_choices* routine performs the following tasks:

(1) Tries to open the groups to list file and calls *email_find_first_non_blank_line* to determine whether the file is empty. If the file is not empty, the data is read from this file until the end of the file. The *strncmp* routine is called to parse the keywords.

(2) If there are no keywords for *EMail* to use, *display_popup_text* is called to display an error message.

(3) If the **groups_to_list** file does not exist, the keyword directory **shared/data/address_groups** is read. If the keyword directory cannot be read, the program exits.

(4) Searches the keyword directory for all selected keywords and puts them into **keyword_list**.

(5) Calls *email_sort_menu_choices* to sort the keyword list.

(6) Calls *email_display_menu_choices* to load the data into the SELECT ADDRESSES menu

(7) Calls *email_display_menu_choices* to load the data into the ADDRESS GROUPS menu

(8) Calls *email_display_menu_choices* to load the data into the default group menu items.

Error Conditions and Handling

Calls *display_popup_text* to display an error message if an error is encountered during processing.

Routine Interfaces

strncmp

display_popup_text

email_sort_menu_choices

email_display_menu_choices

email_find_first_non_blank_line

11.1.58 The email_setup Routine

Purpose

The *email_setup* routine loads a global string used to insert items into the input pad, deselects any selected keywords, initializes the old keyword choices array, resets the message type and address type selection areas, creates the message and address pads, sets the default setup options, and loads a default group if one is found.

Input

global setup options array

Output

message input pad

address input pad

Processing

The *email_setup* routine performs the following tasks:

(1) Checks the keyword list and sets any selected choices to the deselect value.

(2) Creates the message and address pad areas.

(3) Sets the default settings for the setup options.

(4) Sets the extra copy default settings.

(5) Sets the address group name defaults.

(6) Loads the setup options key list.

(7) Builds the setup options adaptation file name and checks whether the file name exists. If it exists, attempts to open the file. If the file cannot be opened, calls *display_popup_text* to display an error message.

(8) Calls *email_find_first_non_blank_line* to find the beginning of the setup options adaptation file.

(9) Gets each option from the setup options adaptation file and checks that its length is greater than **0**, and that the first character is not a # sign. If the first character is a **#** sign, the option is skipped.

(10) Compares the options to the **setup_options_keylist**. If the option is in the keylist, it is added to the **nadin_arinc_data** structure.

(11) If the index is not found and **str** is FALSE, then sets **flag** to FALSE and **flag_set** to TRUE.

(12) Sets **flag_set** to TRUE if index is **8**, **12**, **13**, or **14**.

(13) If index is greater than **0** and **flag_set** is TRUE, then **flag** is set to the selected index option.

(14) If index is greater than **0** and **flag_set** is TRUE, then flag is set to the selected index option.

(15) Sets the extra copy options to FALSE if no node is found.

(16) Sets the CFC extra copy setup options.

(17) Sets the extra copy setup options.

(18) Sets the extra copy node names.

(19) Sets the message type. If there is a numbered advisory message and the ADZY function is enabled, then **email_msg_type** is **advzy**, otherwise the type is **genr**. If advisories are not allowed, the routine forces **email_msg_type** to **genr**.

Error Conditions and Handling

An error message is displayed if the routine is unable to open the **setup** file.

Routine Interfaces

email_create_edit_pad

email_create_ui_msg_header_pad

email_create_addr_pad

display_popup_text

get_word_from_buffer

find_default_group

email_process_type

11.1.59 The email_sort_menu_choices Routine

Purpose

The *email_sort_menu_choices* routine sorts an array of names, which are typically menu choices.

Input

array of menu choices

number of menu choices

Output

sorted array of menu choices

Processing

The *email_sort_menu_choices* routine compares the menu choices and sorts the menu.

Error Conditions and Handling

None

Routine Interfaces

strncmp

strncpy

11.1.60 The find_default_group Routine

Purpose

The *find_default_group* routine loads the addresses for the user's selected default group and sets the group name in the interface.

Input

default group address from setup options **group_to_list** file

Output

displayed addresses that are contained in the default group

Processing

The *email_sort_menu_choices* routine performs the following tasks:

(1) Checks whether the string passed in is a *none* value. If the string is *none*, the routine exits.

(2) Checks the **groups_to_list** file for default group addresses. If the default group is found, gets the default group path. If the default group cannot be found or the **groups_to_list** file could not be opened, *vfmt_$encode* is called to write an error message to **dgroup_err**.

(3) If the default group is not found in the **groups_to_list** file and not **all_groups**, then *vfmt_$encode* is called to write an error message to **dgroup_err**.

(4) If the default group is found, the addresses are loaded into the menus and the group name is set in the user interface.

(5) If the default group is not found and all groups have not been checked, calls *display_popup_text* to display an error message and searches for the default group again.

(6) If the default group is not found and all groups have been checked or the **groups_to_list** file could not be opened, *display_popup_text* is called to display an error message.

Error Conditions and Handling

An error message is displayed if the group is not found in the **groups_to_list** file or if the file could not be accessed.

Routine Interfaces

display_popup_text

email_process_keyword

email_process_apply_address

11.1.61 The move_to Routine

Purpose

The *move_to* routine moves the cursor to the specified place on the *EMail* display.

Input

window stream

x coordinate of the window

y coordinate of the window

Output

Repositioned cursor on the interface

Processing

The *move_to* routine performs the following tasks:

(1) Calls *vfmt_$encode2* to write the specified cursor coordinates to a string **cmd_text**.

(2) Calls *pad_$dm_cmd* to place the cursor at the specified coordinates.

Error Conditions and Handling

None

Routine Interfaces

vfmt_$encode2

pad_$dm_cmd

11.1.62 The move_to_window Routine

Purpose

The *move_to_window* routine moves the cursor to the specified place on the *EMail* display.

Input

window stream

Output

repositioned cursor on the interface

Processing

(1) Calls *pad_$pop_push_window* to pop the desired window in front of all other windows on the display.

(2) Calls *window_coord* to get the top and left coordinates of the window.

(3) Checks the status. If the status is OK then the *move_to* routine is called to move the cursor onto the window.

Error Conditions and Handling

If status from the call to *window_coord* is not zero, the *move_to* routine is not called.

Routine Interfaces

window_coord

move_to

11.1.63 The process_extra_copy_node Routine

Purpose

The *process_extra_copy_node* routine takes the string passed in and places its contents into a copy of the node.

Input

string - contains the nodes for the extra copy option

string length

Output

node name

length of node name

Processing

The *process_extra_copy_node* routine performs the following tasks:

(1) Checks whether the length of the string passed in is greater than 0 and whether the first character is not blank.

(2) If the first and second characters of the string are not a "/", then a "/" is put in the first and second character positions of node and the contents of str are put into node starting at the third character position of node.

(3) If the first and second characters of the string are a "/", then the contents of str are put into node.

Error Conditions and Handling

None

Routine Interfaces

None

11.1.64 The window_coord Routine

Purpose

The *window_coord* routine finds the top and left position of the desired stream.

Input

window stream

Output

top coordinate of window

left coordinate of window

status

Processing

The *window_coord* routine performs the following tasks:

(1) Calls *pad_$inq_windows* to list the windows on the pad.

(2) Sets **top1** to the top of the desired window

(3) Sets **left1** to the leftmost position of the desired window

Error Conditions and Handling

None

Routine Interfaces

pad_$ing_windows

11.2 The email_new_p Module

The *email_new_p* module contains routines related to address processing and dialogue button selection processing. Figure 11-3 shows the task flow of the *email_new_p* routines organization.

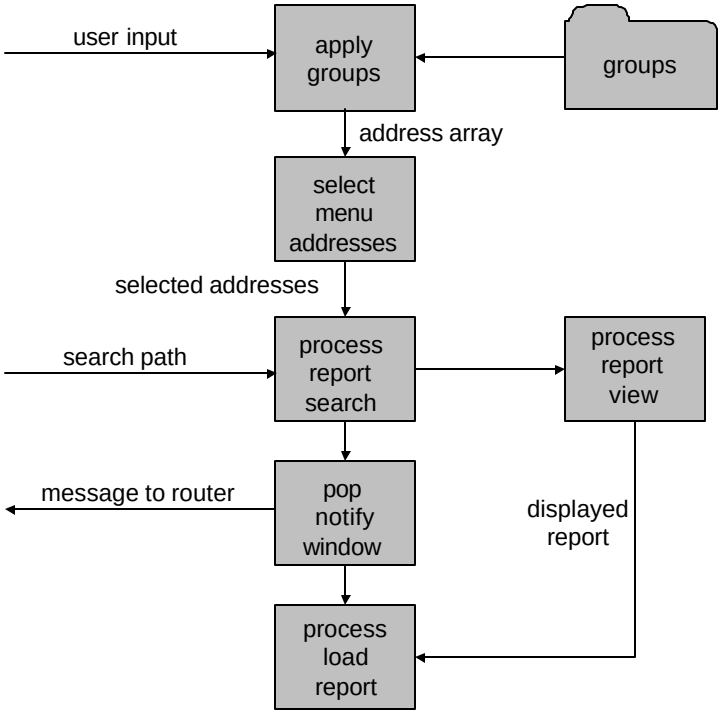

Figure 11-3. Routine Organization for the email_new_p Module

11.2.1 The determine_if_advzy_template Routine

Purpose

The *determine_if_advzy_template* routine checks whether the file being inserted has an advisory header in it and sets the message type to Numbered Advisory if the header is present.

Input

message text file

Output

message header set to numbered advisory

Processing

The *determine_if_advzy_template* routine performs the following tasks:

(1) If **email_msg_type** is **advzy** or ADZY_func is not in the enabled functions file, it returns to the calling procedure.

(2) Calls *email_find_first_non_blank_line* to find the first line of the file.

(3) If it is not the end of the file, puts the advisory header into **advzy_head_str**.

(4) If the header in the file is the same as **advzy_head_str**, sets the message type to Numbered Advisory.

Error Conditions and Handling

None

Routine Interfaces

email_find_first_non_blank_line

11.2.2 The email_apply_keyword_values Routine

Purpose

The *email_apply_keyword_value* routine reads all the address values in the selected keyword file and selects the address in the address array that appears in the selected keyword file.

Input

selected keyword file

index

Output

updated address menu array

Processing

The *email_apply_keyword_value* performs the following tasks:

(1) Gets the full pathname and opens a stream to the file group. If an error occurs, calls *display_popup_text* to display an error message, closes the stream to the pathname, and returns to the calling procedure.

(2) Sets the record length to the size of the record buffer and calls *email_find_first_non_blank_line*, which will find the first line that contains text in the file.

(3) If the record is blank or the end of the file was reached, calls *display_popup_text* to display an error message, closes the stream to the pathname, and returns to the calling procedure.

(4) Calls *get_word_from_buf* to get the key word from the record buffer. If the string length is zero, or if the buffer contains comments, does not increment, removes the new line, and exits.

(5) Classifies the message type as **arn** for ARINC, **aftn** for NADIN, or **apo** for APOLLO. If the message is not any of the three types, searches for a new defined address. If the type cannot be found, the routine will exit. The routine converts the address from actual address to an alias address by calling *email_conv_ifcn_to_ui*.

(6) If the number of names is greater than 1, *email_sort_menu_choices* is called to sort an array of menu names then tries to find the address, which was read into the stored addresses. If the chosen address is not present in the address array, the address is added.

Error Conditions and Handling

Calls *display_popup_text* to display an error message if the type of message cannot be determined.

Routine Interfaces

email_find_first_non_blank_line

display_popup_text

get_word_from_buf

email_conv_ifcn_to_ui

email_sort_menu_choices

11.2.3 The email_clear_search_str Routine

Purpose

The *email_clear_search_str* routine clears the search string entered by the user in the import files popup.

Input

search string entered on the VIEW/IMPORT popup

Output

blank search string

Processing

The *email_clear_search_str* routine calls *dp_$string_set_value* to set **report_keyword_task** to a null string.

Error Conditions and Handling

None

Routine Interfaces

dp_$string_set_value

11.2.4 The email_conv_ifcn_to_ ui Routine

Purpose

The *email_conv_ifcn_to_ui* routine takes the passed address, searches for a matching address in a global list of alternate names, and replaces the address with a corresponding address if one exists.

Input

address

length of address

maximum length of address

alternate address list

Output

address

length of address

Processing

The *email_conv_ifcn_to_ui* routine performs the following tasks:

(1) Determines whether the address that was passed matches an address in **temp^.ifcn_addr_len** file.

(2) If a matching address is found, the routine replaces the passed address with the new corresponding address.

(3) If a matching address is not found, the address is set to the temporary address.

Error Conditions and Handling

None

Routine Interfaces

None

11.2.5 The email_conv_ui_to_ifcn Routine

Purpose

The *email_conv_ui_to_ifcn* routine takes the passed address, searches for a matching address in a global list of alternate names, and replaces the address with a corresponding address if one exists.

Input

address

length of address

maximum length of address

alternate address list

Output

address

length of address

Processing

The *email_conv_ui_to_ifcn* routine performs the following tasks:

(1) Determines whether the address that was passed matches an address in **temp^.ui_addr_len** file.

(2) If a matching address is found, it replaces the passed address with the new corresponding address.

(3) If a matching address is not found, the address is set to the temporary address.

Error Conditions and Handling

None

Routine Interfaces

None

11.2.6 The email_display_gnrl_help Routine

Purpose

The *email_display_gnrl_help* routine displays the general help popup.

Input

dialog popup offset

Output

general help popup displayed on the interface

Processing

The *email_display_gnrl_help* routine performs the following tasks:

(1) Checks that **popup_offset** is **0** and no errors are displayed.

(2) Displays the general help popup if no errors are displayed.

Error Conditions and Handling

None

Routine Interfaces

None

11.2.7 The email_display_menu_choices Routine

Purpose

The *email_display_menu_choices* routine is used to display selected menu choices on a *set_type* menu associated with the passed task identification.

Input

task ID

number of choices

array of choices to display

set_type

Output

displayed menu choices

Processing

The *email_display_menu_choices* routine performs the following tasks:

(1) Gets the menu to be displayed.

(2) Loads the array of menu choices for the menu to be displayed for the set type of menu.

(3) Adds the menu choices and sets the choices to not selected.

(4) Repeats the procedure if an enumerated type of menu is requested.

Error Conditions and Handling

None

Routine Interfaces

None

11.2.8 The email_fill_and_send Routine

Purpose

The *email_fill_and_send* routine creates a file to hold the message, loads the file information into the user message record, and sends the record.

Input

message to send the length of the message text

Output

user message record containing the filename to send

Processing

The *email_display_menu_choices* routine performs the following tasks:

(1) Gets the full pathname of the file. If no error occurs, *transfer_msg_for_send* is called to transfer the file using the pathname.

(2) If an error occurs, *display_st_msg* is called to display an error that the full pathname could not be built to send the file.

Error Conditions and Handling

If an error occurs, *display_st_msg* is called to display an error message to the effect that the full pathname could not be built to send the file.

Routine Interfaces

display_st_msg

transfer_msg_for_send

11.2.9 The email_group_and_send Routine

Purpose

The *email_group_and_send* routine sends the prepared message to all selected destinations.

Input

message to send the length of the message text

Output

user message record transferred to selected addresses

Processing

The *email_group_and_send* routine performs the following tasks:

(1) Gets the current message and increments the message count by one and repeats the send routine until the maximum number of addresses for the specified address type is reached.

(2) Calls *email_conv_ui_to_ifcn* to get the passed address, searches for a matching address in a global list of alternate names, and replaces the address with a corresponding address if one exists.

(3) Calls *email_fill_and_send* to create a file to hold the messages, loads the file information into the user message record, and sends the record. If there are more messages to send, calls *email_fill_and_send* again to send any remaining messages.

Error Conditions and Handling

None

Routine Interfaces

email_conv_ui_to_ifcn

email_fill_and_send

11.2.10 The email_insert_addr_file Routine

Purpose

The *email_insert_addr_file* routine inserts the address file which is related to the saved or sent message selected for restore.

Input

file name

file name length

directory

directory length

Output

restored addresses to the address pad

Processing

The *email_insert_addr_file* routine performs the following tasks:

(1) Determines whether the directory name contains a "/" and encodes it as such. If the directory name does not contain a "/" the routine encodes it without the "/".

(2) Calls *email_file_empty* to determine if the file is empty, and *display_popup_text* is called to display a message to the effect that there are no addresses to restore with the message, and returns to the calling procedure.

(3) Inserts a blank before the restored address and calls *move_to_window* to move the cursor to the specified place in the *EMail* display.

Error Conditions and Handling

If there were no addresses to restore with the message, the *display_popup_text* displays a message and then returns to the calling procedure.

Routine Interfaces

display_popup_text

move_to_window

email_file_empty

11.2.11 The email_insert_file Routine

Purpose

The *email_insert_file* routine inserts the specified file at the beginning of the edit pad.

Input

file name

file name length

directory

directory length

Output

restored message text to the message input pad

Processing

The *email_insert_file* routine performs the following tasks:

(1) Gets the base part of the DM command and sets it equal to the value of text.

(2) Searches for a "/". If one is not present, the routine adds one to the end of the directory.

(3) Gets the text length and the file name length and sets the file name length equal to the value of **text_len**.

(4) Closes the DM command path and calls *move_to_window* to move the cursor to the specified file in the *EMail* display.

Error Conditions and Handling

None

Routine Interfaces

move_to_window

11.2.12 The email_option_keyword_cancel Routine

Purpose

The *email_option_keyword_cancel* routine cancels the group menu and sets the group name back to its previous setting.

Input

group menu cancel task identification

Output

old group name

Processing

The *email_option_keyword_clear* routine performs the following tasks:

(1) Determines whether the *task_id* is equal to *addr_keyword_cancel_task*. If it is, the routine continues to the next step; otherwise, the group menu is not canceled.

(2) Sets all the keyword menu choices to FALSE and sets the current group name to the value of the old group name.

Error Conditions and Handling

None

Routine Interfaces

None

11.2.13 The email_min_max Routine

Purpose

The *email_min_max* routine sets the *EMail* window to minimum and maximum values on the display.

Input

minimum window coordinates

maximum window coordinates

Output

resized *EMail* interface

Processing

The *email_min_max* routine performs the following tasks:

(1) If the window is not at the maximum size, the window height and width are both specified as 1280 and *pad_$set_full_window* is called to establish the window region using the set values.

(2) If the window is at the maximum size, *pad_$set_full_window* is called to reduce the window to the minimum window coordinates using the values stored in the *last_window_size* operation.

Error Conditions and Handling

None

Routine Interfaces

move_to_window

pad_$set_full_window

11.2.14 The email_option_keyword_cancel Routine

Purpose

The *email_option_keyword_cancel* routine cancels the setup option default group menu.

Input

default group menu cancel button task identification

Output

previous default group

Processing

The *email_option_keyword_clear* routine calls *dp_$enum_set_value* to set the current value of 0 to cancel out the menu.

Error Conditions and Handling

None

Routine Interfaces

dp_$enum_set_value

11.2.15 The email_option_keyword_clear Routine

Purpose

The *email_option_keyword_clear* routine clears the setup option default group menu.

Input

default group menu clear button task identification

Output

deselected default groups

Processing

The *email_option_keyword_clear* routine calls *dp_$enum_set_value* to set the current value of **0** for **none** to clear out the menu.

Error Conditions and Handling

None

Routine Interfaces

dp_$enum_set_value

11.2.16 The email_pack_it Routine

Purpose

The *email_pack_it* routine fills the appropriate fields of the user message record in preparation for sending the message.

Input

EMail address type

Output

user message record of type **msg_trans_rec_t**

Processing

The *email_pack_it* routine performs the following tasks:

(1) Determines whether the *EMail* message type is **advzy** and classifies it as ADVY. If the *EMail* message type is not **advzy**, it is classified as GENR.

(2) Determines whether the *EMail* message type is **aftn** and classifies it as a NADIN message and sets the user message destination of NADIN.

(3) Determines whether the *EMail* message type is arnc and classifies it as an ARINC message and sets the user message destination of ARNC.

(4) If the message cannot be classified as any of the above, the user destination is set to APOLLO.

Error Conditions and Handling

None

Routine Interfaces

None

11.2.17 The email_pop_notify Routine

Purpose

The *email_pop_notify* routine pops the notify window from an icon to a full window.

Input

RTR message

Output

ETMS communications message to RTR with type 1903

Processing

(1) Calls *pop_notify* to send a message to *RTR* to pop the Notify window.

(2) If the message cannot be sent, *display_popup_text* displays a message to the effect that the message to pop the Notify window could not be sent.

Error Conditions and Handling

The *email_pop_notify* routine displays an error message if the RTR message could not be sent.

Routine Interfaces

pop_notify

display_popup_text

11.2.18 The email_process_address_selection Routine

Purpose

The *email_process_address_selection* routine updates the array of all address selections.

Input

addresses menu array

Output

updated addresses menu array

Processing

The *email_process_address_selection* routine performs the following tasks:

(1) Gets the types of the addresses and stores them in the array.

(2) Calls *email_read_menu_choices* to convert the TRUE/FALSE values of the address from dialog to **1** or **3** (representing the address types), which are the values used in the array.

Error Conditions and Handling

None

Routine Interfaces

email_read_menu_choices

11.2.19 The email_process_addr_keyword_ok Routine

Purpose

The *email_process_addr_keyword_ok* routine applies the address groups selected from the select addresses popup panel and sets the group name.

Input

keywords

Output

task text for groups

Processing

The *email_process_addr_keyword_ok* routine performs the following tasks:

(1) Determines the number of keywords selected by the user.

(2) If the number of keywords selected is less than one or if the new keyword selected is the same as the old keyword, *email_reset_menu* is called to deselect all choices.

(3) Determines whether there is more than one keyword selected and if so, displays MULTIPLE when the task text is displayed. If there is just one keyword selected, it will display the keyword in the main group area and the popup area.

(4) Calls *email_display_menu_choices* to display selected menu choices on a set-type menu associated with the passed task identification of Apollo, ARINC, or NADIN.

Error Conditions and Handling

None

Routine Interfaces

email_reset_menu

email_display_menu_choices

11.2.20 The email_process_keyword Routine

Purpose

The *email_process_keyword* routine selects/deselects addresses associated with the most recently selected/deselected keyword.

Input

selected keyword

Output

updated menu selections

Processing

The *email_process_keyword* routine performs the following tasks:

(1) Determines what popup, if any, is currently displayed. Sets the *popup_offset* value so the new popup will not be covered by the currently displayed popup.

(2) Determines whether the keyword selected is different from the previously selected keyword, and if so, changes the value of the keyword from selected to deselected.

(3) If a keyword is deselected, *email_apply_keyword_values* is called to ensure that a deselected keyword did not contain an address that was to remain selected.

(4) Sets menus to the correct select/deselect values.

(5) If an error popup is not displayed, the *POPUP_OFFSET* is set back to its previous value.

Error Conditions and Handling

None

Routine Interfaces

email_apply_keyword_values

11.2.21 The email_process_keyword_ok Routine

Purpose

The *email_process_keyword_ok* routine applies the address groups selected from the main panel and sets the group name.

Input

address groups

Output

selected addresses

displayed group name

Processing

The *email_process_keyword_ok* routine performs the following tasks:

(1) Determines the number of keywords selected by the user.

(2) If the number of keywords selected is less than one or if the new keyword selected is the same as the old keyword, *email_reset_menu* is called to deselect all choices.

(3) If the number of keywords selected is greater than one, *email_process_apply_address* is called to process the selection of addresses.

(9) Determines whether there is more than one keyword selected, and if so, displays MULTIPLE when the task text is displayed. If there is just one keyword selected, the keyword displays in the main group area and in the popup area.

Error Conditions and Handling

None

Routine Interfaces

email_reset_menu

email_process_apply_address

11.2.22 The email_process_option_keyword_ok Routine

Purpose

The *email_process_option_keyword_ok* routine sets the default group selected from the setup popup panel and sets the default group name.

Input

address group

Output

selected addresses

displayed group name

Processing

The *email_process_option_keyword_ok* routine performs the following tasks:

(1) Determines whether the number of selected keywords is less than one, and if so, sets the *group_str* to NONE.

(2) If the number of keywords selected is greater than one, gets the string length of the keywords and sets the string to the **default_group_name**.

(3) Displays the default keyword if the number of keywords selected is greater than one and displays NONE if there are no keywords selected.

Error Conditions and Handling

None

Routine Interfaces

None

11.2.23 The email_process_print Routine

Purpose

The *email_process_print* routine closes down the input pad, determines whether the input pad is not empty, invokes a print script to print the contents of the input pad, and redisplays the input pad.

Input

message input pad

Output

contents of the message input pad is printed

Processing

The *email_process_print* routine performs the following tasks:

(1) Calls *display_popup_text* to display a message to the effect that the current message is spooling.

(2) Determines whether the file is not empty, and if it is not empty, calls *email_print_file* to print the message. If the file is empty, calls *display_popup_text* to display a message to the effect that there was no message to print.

(3) Creates a new pad and a window to view it in, sets the window to close when the pad is closed, and defines function keys.

(4) Calls *move_to_window* to move the cursor to the specified place in the *EMail* display.

(5) Returns to the previously stored screen position.

(6) Calls *email_reset_activation_count* to reset the activation count.

Error Conditions and Handling

Calls *display_popup_text* to display a message if a problem was encountered when spooling the message or the message input pad is empty.

Routine Interfaces

display_popup_text

email_print_file

move_to_window

email_reset_activation_count

11.2.24 The email_process_report_choice Routine

Purpose

The *email_process_report_choice* routine displays a report selected from the report popup in the user interface.

Input

selected report from the VIEW/IMPORT popup

Output

displayed report in the message input pad

Processing

The *email_process_report_choice* routine performs the following tasks:

(1) Gets the value of the menu choice and the menu type.

(2) Points to the selected file name and if the "no files" message was selected, returns and sets the task value to **0**.

(3) If the filename is valid, a stream to the file is opened. If an error occurs, *display_popup_text* is called to display a message to the effect that the file cannot be opened.

(4) Determines whether the file type can be opened, and if it cannot be opened, *display_popup_text* is called to display a message to the effect that the type of file cannot be selected.

(5) Saves the position in the report menu, gets the whole file name, and calls *email_insert_file* to insert the file at the beginning of the edit pad. If the

message has an advisory header, it is removed and the cursor is moved to the specified place in the *EMail* display. If the message contains an address file, *email_insert_addr_file* is called to insert the address file that is related to the saved/sent message, which was selected for restore.

(6) Sets the menu task to **0** and calls the **goto_position** to move to the saved position in the reports menu.

(7) If no error message was displayed during processing, calls *email_reset_activation_count* to set the activation count back to **0**, increments the activation count of the **open_popdown_task**, and sets the **popup_offset** to **0**.

Error Conditions and Handling

The *display_popup_text* routine is called to display an error message if the report could not be accessed.

Routine Interfaces

display_popup_text

email_insert_file

email_insert_addr_file

goto_position

email_reset_activation_count

11.2.25 The email_process_report_print Routine

Purpose

The *email_process_report_print* routine displays a report selected from the report popup in a view pad.

Input

selected report

Output

printed report

Processing

The *email_process_report_print* routine performs the following tasks:

(1) If a file is displayed, the file is closed so that another file can be displayed.

(2) Calls *dp_$enum_get_value* to determine which menu task was selected and finds the location of the file.

(3) If the **no files** option is selected, the routine does not load the filename and gets the next file name to be loaded. If an error occurs, *display_popup_text* is called to display an error message to the effect that the file cannot be opened and to check the file choice.

(4) Checks the file type to determine whether the file can be displayed. If an error occurs, the routine first calls *display_popup_text* to display an error message to the effect that the file cannot be printed and then it closes the file stream.

(5) Calls *email_print_file* to print the specified file. If an error occurs, the routine calls *display_popup_text* to display an error message specifying that a problem was encountered in spooling the message and the message will not be printed.

Error Conditions and Handling

If the report cannot be accessed, the following error message is displayed:

The file cannot be opened.

An error message is also displayed if the report cannot be spooled.

Routine Interfaces

email_print_file

dp_$enum_get_value

display_popup_text

11.2.26 The email_process_report_search Routine

Purpose

The *email_process_report_search* routine reads the path name and search values entered in the import files popup and displays the list of available reports in the specified path name matching the search value.

Input

selected search path name

selected search value

Output

displayed list of files matching the search values

Processing

The *email_process_report_search* routine performs the following tasks:

(1) Closes a window if one is being viewed.

(2) Gets the entered path value and converts the path name to lower case ASCII. If the path name contains a "~", the routine gets the full path name. If an error occurs, the routine sets the path and path length to the returned path and path length. Otherwise, **path** is the returned path.

(3) Checks whether the last character in the pathname is a "/". If it is not, a "/" is added to the end of the pathname.

(4) If a node is present or a wildcard *all* is in the pathname for the node, the node name is stripped off for comparison purposes.

(5) If no pathname was entered, the menu type is set to standard message and the menu is cleared.

(6) Determines whether the path selected is a selectable path and marks the file type in the menu.

(7) Searches the string and compares the path to preset types of file types. If the path does not match the preset filetype menu, the file is set to **none selected**.

(8) Converts the file to lower case ASCII. If an error occurs, the search for a string is cleared.

(9) Searches only for files which match the user's selected identification and adds the user identification into the search string. If it is not a user identification search, the routine creates a normal search string.

(10) Displays searching popup on the screen.

(11) If the search is not a date search, the routine calls *filename_search* to display a list of files from the user-entered or selected path matching the user-entered search for string or criteria. If it is a date search, *date_sort_filename_search* is called to display a list of files from the user-entered or selected path matching the user-entered search for string and to sort the files by date.

(12) If an error occurs during processing, *display_popup_text* is called to display an error message.

(13) View/import file menu is cleared.

Error Conditions and Handling

The *email_process_report_search* routine calls *display_popup_text* if an error occurs while searching for the selected files.

Routine Interfaces

filename_search

date_sort_filenaame_search

display_popup_text

11.2.27 The email_process_report_view Routine

Purpose

The *email_process_report_view* routine displays a report selected from the report popup in a view pad.

Input

selected report to view

Output

selected report displayed in the message input pad

Processing

The *email_process_report_view* routine performs the following tasks:

(1) If a file is displayed, the file is closed so another can be displayed.

(2) Calls *dp_$enum_get_value* to determine which menu task was selected and finds the location of the file.

(3) If the **no files** option is selected, the routine does not load the filename and gets the next filename to be loaded. If an error occurs, *display_popup_text* is called to display an error message to the effect that the file cannot be opened and to check the file choice.

(4) Determines whether the file can be displayed by checking the file type. If an error occurs, *display_popup_text* is called to display an error message specifying that the file cannot be viewed.

(5) Calls *pad_$inq_full_window* to get the total display region occupied by the window. If no errors occur, *pad_$inq_window* is called to list the window on the pad. If an error occurs, sets the window size to a window default and creates the new window.

(6) If the window was opened, the routine sets an auto close upon exit and defines function keys.

Error Conditions and Handling

The *display_popup_text* routine is called to display a message if an error occurs while creating a window to the effect that the window size is set to the default value.

Routine Interfaces

display_popup_text

dp_$enum_get_value

pad_$ing_full_window

pad_$ing_window

11.2.28 The email_process_view_on_select Routine

Purpose

The *email_view_on_select* routine displays a report selected from the report popup in a view pad.

Input

selected report

Output

selected report displayed in the message input pad

Processing

The *email_view_on_select* routine determines whether **view_on_select** is activated. If **view_on_select** is activated, *email_process_report_view* is called to display in a view pad a report selected from the report popup.

Error Conditions and Handling

None

Routine Interfaces

email_process_report_view

11.2.29 The email_read_menu_choices Routine

Purpose

The *email_read_menu_choices* routine reads all selected items in the **set_type** menu associated with the passed task identification and adjusts the **ent_type** field in the passed *read_array* to reflect the selections.

Input

task ID

selected items read array

Output

array of read menu choices with converted **ent_type** field

Processing

The *email_read_menu_choices* routine performs the following tasks:

(1) Calls *dp_$set_get_value* to get the task identification.

(2) Converts the TRUE/FALSE values from dialog to **1** or **3** (depending on the menu **set_type**), which are the values used in the array.

Error Conditions and Handling

None

Routine Interfaces

dp_$set_get_value

11.2.30 *The* email_report_cancel Routine

Purpose

The *email_report_cancel* routine clears the selections from the report popup menu.

Input

task identification of the report cancel button

Output

cleared report popup menu

Processing

The *email_report_cancel* routine performs the following tasks:

(1) If a file is displayed, *ios_$close* is called to close the file so that another file can be displayed.

(2) The *dp_$enum_set_value* option is called to set the current value of **report_menu_task** to **0**.

Error Conditions and Handling

None

Routine Interfaces

ios_$close

dp_$enum_set_value

11.2.31 The email_reset_menu Routine

Purpose

The *email_reset_menu* routine deselects all choices on the menu associated with the passed task identification.

Input

task identification for the current menu

Output

deselected menu choices

Processing

The *email_reset_menu* routine performs the following tasks:

(1) If the menu is **set_type**, the routine returns the current value of the task identification and sets the choice to FALSE, which is equivalent to deselected.

(2) If the menu is an enumerated menu type, the routine returns the current value of the task specified and then sets the task value to **0**, which is equivalent to deselected.

Error Conditions and Handling

None

Routine Interfaces

None

11.2.32 The status_popdown Routine

Purpose

The *status_popdown* routine pops down a status message when the OK button is pressed.

Input

task identification status popdown

Output

current popup is removed from the display

Processing

The *status_popdown* routine performs the following tasks:

(1) The *email_reset_activation_count* option is called to set the activation count of the *status_popdown_task* and *popup_offset* back to zero.

(2) Increments by one the activation count of the *status_popdown_task* and *popup_offset*.

(3) If a fatal error occurs, the routine exits and returns to the calling procedure.

Error Conditions and Handling

If a fatal error occurs while removing the popup, the routine will exit.

Routine Interfaces

None

11.2 11.3 The email_m Module

The *email_m* module is the entry point of the program when the program is invoked. The routines that comprise *email_m* contain the main event loop and dialog initializing call.

11.3.1 The check_window_size Routine

Purpose

The *check_window_size* routine checks the current window size and resizes the window if it is below the minimum size or above the maximum size.

Input

current window size **x**,**y** height and width display coordinates

Output

resized *EMail* interface

Processing

The *check_window_size* routine performs the following tasks:

(1) Gets the current screen size and determines whether it is smaller than the minimum size of 700 for height and 1130 for width.

(2) If the current screen size is above the maximum size, the screen is set to the maximum window size. The *pad_$inq_full_window* is called to display the total display region occupied by the window.

(3) If the current screen size is smaller than the minimum size, the screen is set to the minimum window size. The *pad_$inq_full_window* is called to display the total display region occupied by the window.

(4) Stores the value of the window size in **last_window_size**.

Error Conditions and Handling

None

Routine Interfaces

pad_$ing_full_window

11.3.2 The email_init_dialog Routine

Purpose

The *email_init_dialog* routine creates a pad for dialog to run and initializes dialog to run in the pad.

Input

transcript pad for the dialog to run in the dialog stream

Output

activation count of the active task

Processing

The *email_init_dialog* routine performs the following tasks:

(1) Creates a transcript pad at the right side of the existing window and opens the dialog stream to the pad.

(2) Initializes the dialog stream from memory.

(3) Increments by one the activation count of the active task group.

Error Conditions and Handling

If the routine is unable to initialize the dialog, an error message is written to the standard output and the process exits.

Routine Interfaces

None

11.3.3 The force_dialog_display Routine

Purpose

The *force_dialog_display* routine is called to force the drawing of the dialog interface before the open call to the node switch blocks.

Input

dialog stream identification

Output

displayed dialog interface

Processing

The *force_dialog_display* routine calls *dp_$cond_event_wait* to draw the dialog interface before blocking the open call.

Error Conditions and Handling

None

Routine Interfaces

dp_$cond_event_wait

11.3.4 The get_screen_size Routine

Purpose

The *get_screen_size* routine gets the size in pixels of the current screen.

Input

current display

Output

x,y display size coordinates

Processing

The *get_screen_size* routine performs the following tasks:

(1) Gets the size of the display of the current screen by callin g *gpr_$ing_disp_characteristics* and assigns the size to **x** and **y** variables.

(2) If an error occurs, uses the default size of **x**=1024 and **y**=800.

Error Conditions and Handling

If an error occurs, uses the default size of **x=1024** and **y=800**.

Routine Interfaces

gpr_$ing_disp_characteristics

11.3.5 The process_dialog_events Routine

Purpose

The *process_dialog_events* routine is called when a dialog event is triggered. The event is read and processed.

Input

dialog event count

Output

processed event

Processing

The *process_dialog_events* routine performs the following tasks:

(1) Gets the event counter of the dialog event.

(2) If an error occurs, *display_st_msg* is called to display a message to the effect that there was an error from the dialog event wait. The routine then sets the task to **0** and then exits the process.

(3) Otherwise, the dialog is read and processed.

Error Conditions and Handling

If an error occurs, *display_st_msg* is called to display a message that there was an error from the dialog event wait, sets the task to **0**, and then exits the process.

Routine Interfaces

display_st_msg

11.3.6 The main program

Purpose

The main program calls all other routines to run *EMail*.

Input

program invocation

Output

EMail user interface

Processing

The main program performs the following tasks:

(1) *Check_window_size* is called to check the current window size and resize the window if it is below the minimum size or above the maximum size.

(2) *EMail_init_dialog* is called to create a pad for dialog to run and to initialize dialog to run in the pad.

(3) *Setup_dp_event* is called to establish the dialog event counter value and the event counter pointer. If an error occurs, displays a message specifying that the dialog event could not be set.

(4) *Force_dialog_display* is called to force the drawing of the dialog interface before the open call to the node switch blocks. *Display_popup_text* is called to display a message to the user that *EMail* is initializing.

(5) *EMail_init* is called to perform initialization for *EMail* processing.

(6) Logical name is loaded with the node name and *email_reset_activation_count* is called to set the activation count of the task back to **0**.

(7) *Display_popup_text* is called to do the following

- o Display a message to the effect that *EMail* is trying to connect to a node switch
- *o* Open the communication channel calling *open_com*
- **o** Reset the activation count to **0**

(8) If the status message is a fatal error message, then the event counter is set to **1**. If the status message is not a fatal error message, the event counter is set to the value of **max_ec**.

(9) Increments the event count wait as needed. If an error occurs, the routine first calls *display_st_msg* to display a message specifying that there was an error from *ec2_wait*, and then exits the process.

(10) Initializes the dialog event counter and the communication event counter. Checks *RTR* status and calls *reset_events* to reset the event counter value and pointer.

(11) If an error occurs, *display_st_msg* is called to display an error message or a fatal error message. If the status message is not a fatal message, the connection status is correct, the communication channel is open, and the *RTR* is alive. The routine then determines the size of what is connected and connects the current task. Otherwise, if the status message is not fatal, the routine gets the size of what is being connected and connects the current task.

(12) *Check_window_size* is called to check the current window size and resize the window if the window is below the minimum size.

(13) When done, *email_close_up_shop* is called to prepare for the termination of *EMail* by deleting the input pads, closing the communication channel, and then exiting the program. If an error occurred, the routine calls *email_close_up_shop* again to terminate *EMail*.

Error Conditions and Handling

If an error occurs during processing, *display_st_msg* and *display_popup_text* are called to display the appropriate error message. If the error was fatal, *email_close_up_shop* is called and the program exits.

Routine Interfaces

check_window_size

email_init_dialog

setup_dp_event

force_dialog_display

email_init

display_popup_text

email_reset_activation_count

open_com

reset_events

display_st_msg

email_close_up_shop

11.4 EMail Data Structure Tables

Tables in sections 11-1 through 11-8 describe data structures used by *EMail*. Included are descriptions of data structures internally used by *EMail* and those used for communicating between this and other processes. The tables describe each data item within the data structures. This description includes a definition, unit/format, range, and variable type. When the unit/format field or range field is blank, this indicates that a specific unit/format or range was not defined.

11.4.1 The alt_addr_rec Data Structure

The *alt_addr_rec* data structure is used by the *email_read_alt_addr* routine. Alternate names for selected addresses found in /etms5/email/data/address_files/address_mappings are stored in this data structure to be presented in the user interface.

Table 11-1. alt_addr_rec Data Structure

alt_addr_rec				
Library Name: Email_lib		**Purpose:** Alternate address record definition		
Element Name: email_types.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
ifcn_addr	ifcn address			string
ui_addr	user interface address			string
ifcn_addr_len	ifcn address length			integer
ui_addr_len	user interface address length			integer
next	next address			alt_addr_list_t

11.4.2 The long_text_rec_t Data Structure

The *long_text_rec_t* data structure is used by the *email_fill_and_send* routine. The last message built is stored in this data structure, the file information is loaded into the user message record, and the message is sent.

Table 11-2. long_text_rec_t Data Structure

long_text_rec_t				
Library Name: Email_lib		**Purpose:** Long text record type definition		
Element Name: email_types.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
text	message text			name_$lon_pname_t
size	message text size			integer

11.4.3 The setup_options_key_rec_t Data Structure

The *setup_options_key_rec_t* data structure is used by the *email_setup* routine. Setup options are stored in this data structure to be loaded into the *EMail* user interface menus.

Table 11-3. setup_options_key_rec_t data structure

setup_options_key_rec_t				
Library Name: Email_lib		**Purpose:** Setup options key record definition		
Element Name: email_types.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
option_name	option name			string
option_index	index of option			integer

11.4.4 The email_pgm_arg Data Structure

The *email_pgm_arg* data structure is used by the *print_file_at_node* and *email_print_file* routines. Print commands are stored in this data structure and are invoked by the *email_print file* routine to print the message pad or by the *print_file_at_node* routine to copy the file to another node's print queue.

Table 11-4. email_pgm_arg Data Structure

email_pgm_arg				
Library Name: Email_lib		**Purpose:** Email program argument record definition		
Element Name: email_p.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
len	length			integer
chars	print command name			name_$long_pname_t

11.4.5 The node_record Data Structure

The *node_record* data structure is used by the *email_init* routine. This data structure stores a list of the nodes on the network.

Table 11-5. node_record Data Structure

setup_options_key_rec_t				
Library Name: common		**Purpose:** Node record definition		
Element Name: email_types.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
name	node name			string
name_len	node name length			integer
time_offset	time offset			integer32

11.4.6 The file_record Data Structure

The *file_record* data structure is used by EMail routines to store the files to be processed.

Table 11-6. file_record Data Structure

file_record				
Library Name: common_includes		**Purpose:** File record definition		
Element Name: email_ats_types.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
name	file name			string
name_len	node name length			integer
time_offset	time offset			integer32
stream	data stream			stream_$id_t
good	good file indicator		true or false	boolean

11.4.7 The nadin_arinc_rec_t Data Structure

The *nadin_arinc_rec_t* data structure is used by the *email_pack_it* routine. The user message data is stored in this data structure in preparation for sending the message.

Table 11-1. alt_addr_rec Data Structure

alt_addr_rec				
Library Name: common_includes		**Purpose:** NADIN, ARINC record definition		
Element Name: email_ats_types.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
aftn_priority	ariority of aftn			text_rec_t
aftn_source	aftn source			text_rec_t
arnc_priority	arnc priority			text_rec_t
arcn_source	arnc source			text_rec_t
phone_number	term phone number			text_rec_t
atcc_var	advisory prefix			text_rec_t

11.4.8 The user_msg_t Data Structure

The *user_msg_t* data structure is used by *EMail* to implement a connection to the *etms5* communication toolkit.

Table 11-8. user_msg_t Data Structure

user_msg_t				
Library Name: common_includes		**Purpose:** User message type definition		
Element Name: new_com_types.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
from	process name of message sender			char6_t
addresses	destination addresses			address_field_t
origin	origin record			char8_t
tail	phone number text			char69_t
no_addr	number of destination addresses			integer
msg_dest	destination of message			char6_t
msg_type	type of message			char4_t
msg_len	length of filename of blank			integer
msg_code	message code	file or message indiator	0 to 1	char

11.5 EMail Cross Reference List

The following list can be used to locate routines within the *EMail* section.

Routine ..... Section

break_down_node_list 11.1.1

build_global_paths 11.1.2

check_nadin_arinc_data 11.1.3

check_window_size 11.3.1

copy_list 11.1.4

determine_addr_type 11.1.5

determine_if_advzy_template 11.2.1

display_st_msg 11.1.6

Routine ..... Section

display_popup_text 11.1.7

*email_addr*_cancel 11.1.8

email_apply_keyword_values 11.2.2

email_break_up_line 11.1.9

email_check_on_quit 11.1.10

email_clear_address_array 11.1.11

*email_clear_addrs_* 11.1.12

email_clear_addrs_pad 11.1.13

email_clear_search_str 11.2.3

email_close_up_shop 11.1.14

email_conv_ifcn_to_ui 11.2.4

email_conv_ui_to_ifcn 11.2.5

email_create_address_pad 11.1.15

email_create_edit_pad 11.1.16

email_create_ui_addr_header_pad 11.1.17

email_create_uimsg_header_pad 11.1.18

email_delete_addr_pad 11.1.19

email_delete_edit_pad 11.1.20

email_display_gnrl_help 11.2.6

email_display_menu_choices 11.2.7

email_extract_body_of_message 11.1.21

email_file_empty 11.1.22

email_fill_and_send 11.2.8

email_find_first_non_blank_line 11.1.23

email_generate_advisory_number 11.1.24

Routine ..... Section

email_get_args 11.1.25

email_get_user_id 11.1.26

email_get_word 11.1.27

email_group_and_send 11.2.9

email_init 11.1.28

email_init_dialog 11.3.2

email_insert_addr_file 11.2.10

email_insert_file 11.2.11

email_keyword_cancel 11.2.12

email_make_contains_file_list 11.1.29

email_make_display_file_list 11.1.30

email_make_icon 11.1.31

email_make_save_name 11.1.32

email_min_max 11.2.13

email_option_keyword_cancel 11.2.14

email_option_keyword_clear 11.2.15

email_pack_it 11.2.16

email_parse_func_arg 11.1.33

email_pop_notify 11.2.17

email_print_file 11.1.34

email_print_file_at_node 11.1.35

email_process_address_pad 11.1.36

email_process_address_selection 11.2.18

email_process_addr_keyword_ok 11.2.19

email_process_apply_address 11.1.37

Routine ..... Section

email_process_check_print 11.1.38

email_process_check_send 11.1.39

email_process_file_type 11.1.40

email_process_keyword 11.2.20

email_process_keyword_ok 11.2.21

email_process_options 11.1.41

email_process_option_keyword_ok 11.2.22

email_process_print 11.2.23

email_process_report_choice 11.2.24

email_process_report_print 11.2.25

email_process_report_search 11.2.26

email_process_report_view 11.2.27

email_process_reset 11.1.42

email_process_save 11.1.43

email_process_send 11.1.44

email_process_type 11.1.45

email_process_view_on_select 11.2.28

email_quit_confirm 11.1.46

email_read_alt_addr 11.1.47

email_read_menu_choices 11.2.29

email_remove_duplicate_menu_choices 11.1.48

email_report_cancel 11.2.30

email_reset_activation_count 11.1.49

email_reset_groups 11.1.50

email_reset_menu 11.2.31

Routine ..... Section

email_save_addrs 11.1.51

email_save_over 11.1.52

email_save_over_no 11.1.53

email_save_cancel 11.1.54

email_select_all_groups 11.1.55

email_set_address_choices 11.1.56

email_set_keyword_choices 11.1.57

email_setup 11.1.58

email_sort_menu_choices 11.1.59

find_default_group 11.1.60

force_dialog_display 11.3.3

get_screen_size 11.3.4

main 11.3.6

move_to 11.1.61

move_to_window 11.1.62

process_dialog_events 11.3.5

process_extra_copy_node 11.1.63

status_popdown 11.2.32

window_coord 11.1.64