BBN Systems and Technologies
A Division of Bolt Beranek and Newman Inc.

BBN Report No. 7628

SIMNET CVCC

Radio Performance Monitor (RADMON) CSCI

Volume I

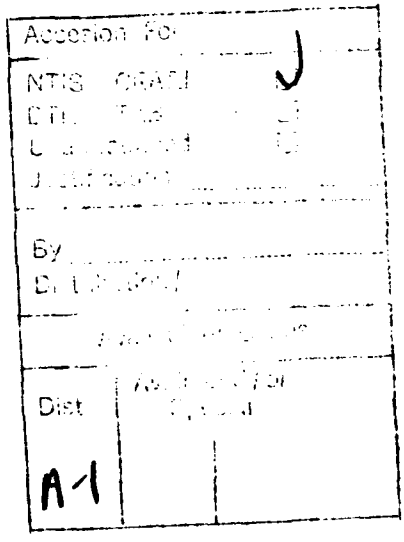
Software Design Document

SIMNET CVCC

Radio Performance Monitor (RADMON) CSCI

Volume I

Software Design Document



Contract No. MDA972-89-C-0060
Contract No. MDA972-90-C-0061
CDR Sequence No. 0002AB
December 1991

Prepared by:
Bolt Beranek and Newman Inc.
Systems and Technologies
Advanced Simulation
10 Moulton Street
Cambridge, MA 02138 USA

Prepared for:
Defense Advanced Research Projects Agency (DARPA)
3701 North Fairfax Street
Arlington, VA 22203-1714

This research was performed by BBN Systems and Technologies under Contract Nos. MDA972-89-C-0060 and MDA972-90-C-0061 to the Defense Advanced Research Projects Agency (DARPA). The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policy, either expressed or implied, of DARPA, the U.S. Army, or the U.S. Government.

Table of Contents

1. SCOPE

This Software Design Document (SDD) establishes the design for the Computer Software Configuration Item (CSCI) identified as the Radio Perfomance Monitor (Radmon) of the Simulation Network (SIMNET) system.

This SDD specifies the preliminary design for the Radmon CSCI. This includes descriptions of external interfaces and Computer Software Component (CSC) decomposition.

Section 2., Referenced Documents, lists documents referenced in this SDD.

Section 3., CSCI Design, provides an overview of this CSCI: its role, interfaces with the rest of the SIMNET system, and its internal organization. We also describe the individual CSCs that comprise this CSCI in terms of execution control and data flow, internal interfaces, and relationships among CSCs.

This CSCI was developed according to the following software standards: X-Window System, version 11, (MIT 1987), and OSF/Motif (Open Software Foundation).

2. REFERENCED DOCUMENTS

The following documents are referenced by this Software Design Document.

2.1 Government Documents

None

2.2 Non-Government Documents

2.2.1 Specifications

None

2.2.2 Standards

None

2.2.3 Drawings

None

2.2.4 Other Publications

SIMNET CVCC Analysis of Simulated SINCGARS Communications in the SIMNET CVCC Company-Level Experiments: Ft. Knox Close Combat Testbed, January-May, 1990, BBN Report No. 7690 (May 1991).

SIMNET CVCC Simulation of the SINCGARS Radio System Software Design Document , BBN Report No. 7632 (July 1991).

SIMNET Simulation of Radio Communication: A Testbed for Investigating Combat Vehicle C^3I Technology, BBN Report No. 7352 July 1990); (includes *Appendix C: Radio Performance Monitor*).

3. CSCI DESIGN

3.1 CSCI Overview

The Radio Performance Monitor (Radmon) CSCI is used to monitor and analyze the performance of the SINCGARS radio simulation for SIMNET. Radmon monitors traffic on the simulation network and records information broadcast by the SINCGARS radio simulator hosts concerning every important state change of every simulated radio. Radmon maintains a running history of all state changes for each radio.

The information that Radmon records can be saved in a file and subsequently restored from that file. Radmon allows the user to browse through recorded data to examine interactions in detail. The recorded data can also be summarized to show channel utilization, radio utilization, and other statistics of radio activity.

The collected information can be presented in several ways:

- A stripchart display shows the activity of radios as a function of time.
- A connectivity map shows the communication paths that are usable.
- A status display shows the detailed state of individual radios.

The Radmon CSCI interfaces externally to the SINCGARS radio simulation hosts. These simulation hosts monitor the position of various knobs and buttons on simulated radios and broadcasts "transmitter PDUs" reflecting the state information about each radio. They also monitor "vehicle appearance PDUs" broadcast by vehicle simulators to determine the position of the vehicles and hence the radios in those vehicles. The radio simulation host broadcasts a "receiver PDU" each time the signal a radio is receiving changes. The Radmon CSCI records information broadcast by the radio simulator hosts.

Radmon is an X-windows application using the *Motif* toolkit. The content and layout of the user interface is specified using *Motif UIL* (User Interface Language). An X-windows resource file specifies details of the interface such as button labels. In addition to X-windows events, Radmon uses the UNIX alarm clock signal to periodically wake up and process PDUs received from the simulation network.

3.1.1 CSCI Architecture

Radmon functionality can be broken down into four top-level CSCs, shown in Figure 3-1:

- Top-Level Radio Monitor Software
- Data Acquisition
- Graphical User Interface (GUI)
- Utilities

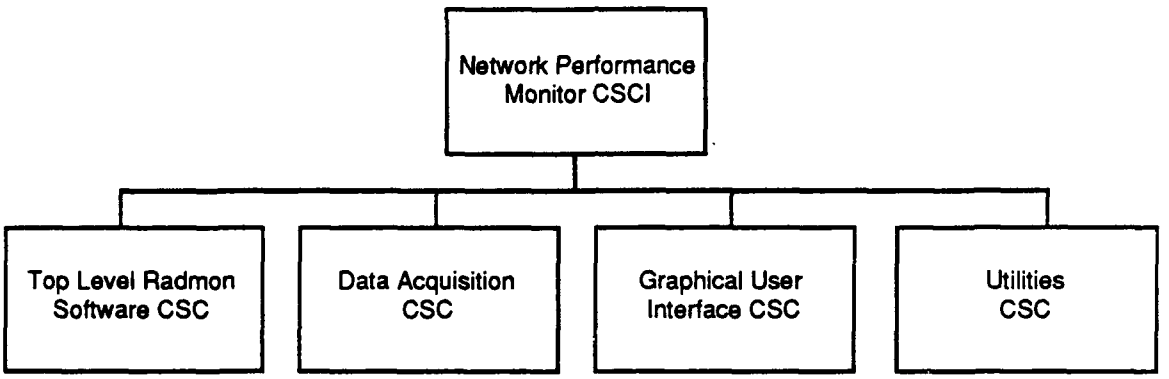


Figure 3-1. Radmon CSCI Structure

The Top Level Radmon Software CSC provides the program entry point, and the main simulation loop is found here. This CSC initializes tune and state routines and the network interface; it also invokes routines to select the mode, and timer routines to update the map display and generate vehicle and radio names. It consists of two second-level CSCs: Radmon and Display.

The Data Acquisition (DA) CSC captures the exchange of data between radio simulation hosts. It records this exchange as a time-stamped sequence of events, which in turn serves as raw data for the Graphical User Interface CSC. The data can be used in real time or stored in a file and retrieved later.

The GUI CSC sorts out the raw data from the DA CSC and puts it into a format that can be used by the operator to display connectivity, specific parameters for specific radios, operating status of radios, and the activity of radios as a function of time.

The Utilities CSC provides routines used by the other CSCs. These include hash table routines, routines for manipulating time lines, and routines to keep track of tunings.

3.1.2 System States and Modes

At the top level, the Radmon CSCI operates in two modes: on-line and off-line. In on-line mode, data acquisition occurs and live data is displayed. In off-line mode, there is no real-time data acquisition; instead, data obtained from stored files is displayed.

3.1.3 Memory and Processing Time Allocation

Memory and processing time are allocated to the Data Acquisition CSC as needed.

3.2 CSCI Design Description

3.2.1 Top-Level Radmon Software CSC

The Top Level Radmon Software CSC provides the program entry point, and the main simulation loop is found here. This CSC initializes tune and state routines and the network interface; it also invokes routines to select the mode, and timer routines to update the map display and generate vehicle and radio names. It consists of two second-level CSCs: radmon and display, as illustrated in Figure 3.2.1-1.

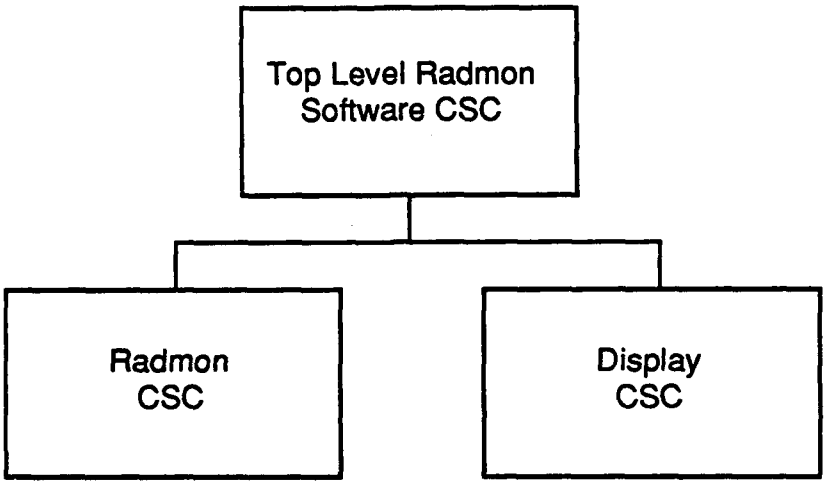


Figure 3.2.1-1. Top-Level Radmon Software CSC

3.2.1.1 Radmon CSC

The Radmon CSC provides the program entry point, and the main simulation loop is found here. This CSC initializes tune and state routines and the network interface; it also invokes routines to select the mode, update the map display, generate vehicle and radio names. The Radmon CSC is comprised of the CSUs shown in Figure 3.2.1.1-1.

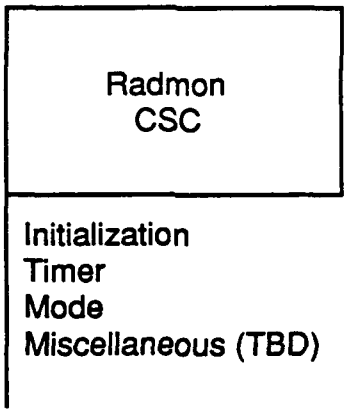


Figure 3.2.1.1-1. Top Level Software—Radmon CSC Structure

Initialization CSU

The Initialization CSU consists of routines that initialize the Radmon CSC.

main

main provides the program entry point for the CSCI, and the main loop is executed here. It announces the program name, sets up the system to catch normal termination signals, processes command and line arguments, debugs the system as necessary, and initializes the display, tune and state routines, and the network interface. Its call is int main (argc, argv).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
argc	int	See Appendix.
argv	pointer to char	See Appendix.

Return Values		
Return Value	**Type**	
0	int	

Calls	
Function	**Where Described**
display_init(argc, argv)	display.c
htReplenish()	hTable.c
htFreeInit()	hTable.c
InitSimNetwork()	net.c
netReplenish()	net.c
resetTime()	radmon.c
Online()	radmon.c
pint_banner()	radmon.c
stateReplenishFreeList()	radmon.c
state_init()	state.c
tlReplenishFreeList()	timeline.c
tuneReplenishFreeList()	tune.c
tune_init()	tune.c

InitVehicle

InitVehicle initializes a vehicle table entry. It is called as net_vehicles grows to initialize new entries. The function call is void InitVehicle(vp, vidx).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vp	VehicleInfoP	See Appendix.
vidx	int	See Appendix.

Calls	
Function	**Where Described**
state_find(st)	state.c

set_identification_label

set_identification_label sets the identification label widget to identify the data currently being presented. This may be data from a file or it may be on-line currently being acquired from the simulation network. The function call is void set_identification_label().

Calls	
Function	**Where Described**
set_widget_label(va_alist)	status.c

DestroyEverything

DestroyEverything calls all the destroy routines to release all the resources associated with the current data. The function call is void DestroyEverything().

Calls	
Function	**Where Described**
DestroyVehicles()	radmon.c
state_destroy()	state.c
traffic_destroy()	traffic.c
tune_destroy()	tune.c
display_deselect()	display.c
map_destroy()	map.c
net_destroy()	net.c

DestroyWidget

DestroyWidget destroys a widget, insuring that the widget exists and that the point is nullified afterward. The function call is void DestroyWidget(wp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
wp	pointer to Widget	See Appendix.

DestroyVehicles

DestroyVehicles reinitializes the net_vehicles to its initial state, and resources are released. The function call is void DestroyVehicles().

Calls	
Function	**Where Described**
DestroyWidget()	radmon.c
tlFree(rp)	timeline.c

resetRadios

resetRadios discards state information for all radios. The radios are otherwise retained. The function call is void resetRadios().

Calls	
Function	**Where Described**
netChangeState(rp, st, Ttimestamp)	net.c
tlFree(rp)	timeline.c

vehicle_name

vehicle_name generates a vehicle name. A vehicle is named <battalion><company><bumper> if that information is known; otherwise, it is named with its vehicle ID. The function call is char *vehicle_name(vp).

Return Values		
Return Value	**Type**	**Meaning**
buf	char	Vehicle name

radio_name

radio_name generates the name of a radio. A radio is named <vehicle name>/[A|B] depending on whether this is the first or second radio in the vehicle. The function call is char *radio_name(rp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
buf	char	Radio name

Calls	
Function	**Where Described**
vehicle_name(vp)	radmon.c

state_radio_name

state_radio_name generates the name of the radio which is transmitting in a particular state. The function call is char *state_radio_name(st).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
st	STATEP	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
radio_name(net_radios[st->st_transmitter])	char	Radio name

Calls	
Function	**Where Described**
radio_name(rp)	radmon.c

print_banner

print_banner prints the banner :

SINCGARS RadioMonitor
%s
BBN Systems and Technologies Corporation
Cambridge, MA, 02138

The function call is void print_banner().

Timer CSU

The Timer CSU consists of routines that update displays and control processing intervals.

slow_periodic

slow_periodic is a timer routine to update things such as the various displays about every second. It also replenishes resources used by AST level code. The function call is void slow_periodic().

Calls	
Function	**Where Described**
htReplenish()	hTable.c
map_advance_display(up_to, max_advance, if_frozen)	map.c
netReplenish	net.c
EnablePeriodic()	radmon.c
TimeoutRadios()	radmon.c
TimeoutVehicles()	radmon.c
stateReplenishFreeList()	state.c
state_update()	state.c
tlReplenishFreeList()	timeline.c
UpdateRadioTraffic()	traffic.c
tuneReplenishFreeList()	tune.c

medium_periodic

medium_periodic is a timer routine called every 1/4 second to advance the map display. The function call is void medium_periodic().

EnablePeriodic

EnablePeriodic turns the fast periodic processing back on. The function call is void EnablePeriodic().

DisablePeriodic

DisablePeriodic turns the fast periodic processing off. The function call is void DisablePeriodic().

fast_periodic

fast_periodic is invoked frequently via an alarm clock signal to process incoming PDUs. The function call is void fast_periodic().

Calls	
Function	**Where Described**
ReadPDUs()	net.c
EnablePeriodic()	radmon.c

resetTime

resetTime resets the simulation clock time to zero. The function call is resetTime().

Mode CSU

Offline

Offline performs various actions to switch the monitor from online mode to offline mode. The function call is void Offline().

Calls	
Function	**Where Described**
map_reset()	map.c
set_identification_label()	radmon.c
traffic_reset()	traffic.c

Online

Online performs various actions to switch the monitor from offline mode to online mode. The function call is void Online().

Calls	
Function	**Where Described**
map_reset()	map.c
netgeteaddr()	net.c
set_identification_label()	radmon.c

Miscellaneous—CSU category to be determined

TimeoutRadios

TimeoutRadios marks radios for which no transmitter PDU has been received for 12 seconds as non-existent. The function call is void TimeoutRadios().

Calls	
Function	**Where Described**
netChangeState(rp, st, timestamp)	net.c

TimeoutVehicles

TimeoutVehicles marks vehicles for which no appearance PDU has been received for 12 seconds as non-existent. The function call is void TimeoutVehicles().

Calls	
Function	**Where Described**
map_vehicle_vanished(vp1)	map.c

exit_gracefully

exit_gracefully cleans up the screen before exiting. The function call is void exit_gracefully().

3.2.1.2 Display CSC

The Display CSC contains routines related to the top-level display widgets. It is comprised of three CSUs, shown in Figure 3.2.1.2-1.

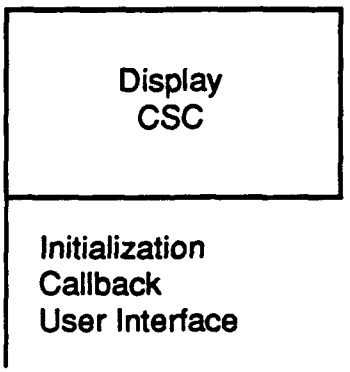


Figure 3.2.1.2-1. Top Level Software—Display CSC Structure

Initialization CSU

function_descend

function_descend descends through selected buttons of the button list tree applying fn to the button lists. The function call is void function_descend(fbc, fn).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fbc	FUNCTION_BUTTON_CLOSUREP	See Appendix.
*fn	void	See Appendix.

create_function_items

create_function_items constructs a resource list and retrieves the label strings to be put on the buttons. It then constructs the item lists for the button lists and recursively does the same for sub-function button lists. The function call is void create_function_items(w, fbc).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
fbc	FUNCTION_BUTTON_CLOSUREP	See Appendix.

NameToWidget

NameToWidget gets a widget id for the named child of a widget. It checks for failure and aborts. The function call is Widget NameToWidget(parent, name).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	See Appendix.
name	pointer to char	See Appendix.

Return Values		
Return Value	**Type**	
widget	Widget	

CvtStringToWidget

CvtStringToWidget is a resource converter from string to widget, where the string is the name of the widget relative to the parent. The function call is void CvtStringToWidget (args, nargs, fromVal, toVal).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
args	XrmValuePtr	See Appendix.
nargs	pointer to int	See Appendix.
fromVal	XrmValuePtr	
toVal	XrmValuePtr	

Calls	
Function	**Where Described**
NameToWidget	display.c

CvtStringToStringLtoR

CvtStringToStringLtoR converts a string to a compound converting \n to separators. The function call is void CvtStringToXmStringLtoR (args, nargs, fromVal, toVal).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
args	XrmValuePtr	See Appendix.
nargs	pointer to int	See Appendix.
fromVal	XrmValuePtr	See Appendix.
toVal	XrmValuePtr	See Appendix.

CvtStringToColor

CvtStringToColor converts a string to an XColor. The function call is void CvtStringToColor (args, nargs, fromVal, toVal).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
args	XrmValuePtr	See Appendix.
nargs	pointer to int	See Appendix.
fromVal	XrmValuePtr	See Appendix.
toVal	XrmValuePtr	See Appendix.

display_init

display_init initializes lots of display stuff and creates most of the widget tree. The function call is void display_init(argcp, argv).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
argcp	pointer to int	See Appendix.
argv[]	*array of pointer to char*	See Appendix.

Calls	
Function	**Where Described**
create_function_items(w, fbc)	display.c
NameToWidget(parent, name)	display.c
fileRegisterNames()	file.c
fileInitWidgets()	file.c
mapRegisterNames()	map.c
mapInitWidgets()	map.c
stateRegisterNames()	state.c
stateInitWidgets()	state.c
trafficRegisterNames()	traffic.c
trafficInitWidgets()	traffic.c

Callback CSU

The Callback CSU consists of functions that activate callbacks for various buttons.

quit_callback

quit_callback activates the callback for the quit button and exits the program. The function call is void quit_callback(w).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.

Calls	
Function	**Where Described**
exit_gracefully	radmon.c

dump_callback

dump_callback activates the callback for the dump button and dumps the widget tree to stdout for debugging. The function call is void dump_callback(w).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.

Calls	
Function	**Where Described**
EnablePeriodic()	radmon.c
DisablePeriodic()	radmon.c

reset_callback

reset_callback activates the callback for the reset button. It returns to online mode if necessary and deletes all recorded data. The function call is void reset_callback(w).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.

Calls	
Function	**Where Described**
map_reset()	map.c
DrainPDUs()	net.c
EnablePeriodic()	radmon.c
DisablePeriodic()	radmon.c
resetTime()	radmon.c
DestroyEverything()	radmon.c
Online()	radmon.c
resetRadios()	radmon.c
traffic_reset()	traffic.

function_callback

function_callback activates the callback for the function button list. If the selected function has sub-functions, the current sub-function widgets are unmanaged. The sub-function widget is activated and any previously active sub-sub-function widgets are reactivated. Then the function attached to the button is executed. The function call is void function_callback(w, fbc, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
fbc	FUNCTION_BUTTON_CLOSUREP	See Appendix.
cback	pointer to XMButtonLCallbackStruct	See Appendix.

Calls	
Function	**Where Described**
function_descend	display.c

User Interface CSU

The User Interface CSU consists of routines that control the user interface.

display_deselect

display_deselect deselects the selected button, if any. The function call is void display_deselect().

clipResize

clipResize resizes the callback for the clipping widget of the general_area_widget. It attempts to make the width of the Pack widget track the clipping widget. There are scenarios in which this can fail. It's not clear if these failures can be corrected given the way the X toolkit behaves. The function call is void clipResize(w, data, cback).

Parameters		
Parameter	Type	Where Typedef Declared
w	Widget	See Appendix.
data	caddr_t	See Appendix.
cback	XmAnyCallbackStruct	See Appendix.

Dispatch

Dispatch disposes of all X events. The function call is void Dispatch().

3.2.2 Data Acquisition CSC

This CSC monitors transmitter and receiver PDUs broadcast by the radio simulation hosts and the vehicle appearance PDUs broadcast by vehicle and radio simulation hosts. As each PDU is received, the state of the affected radio or radios is changed. A radio's state includes the following information:

- Tuning
- Status (non-existent, inactive, inoperative, receiving, not-receiving, transmitting)
- Speaker (if status equals transmitting)
- Antenna height
- Received power (if receiving or not-receiving)

- Who is transmitting (if receiving or not-receiving)
- Geographic position

The new state is found in or entered into a hash table and the pointer to that hash table entry, along with the time at which the PDU was received, is appended to the timeline for that radio. A separate timeline is maintained for each radio. As new radios appear on the network, new timelines are created. All timelines are maintained in main memory. The timelines comprise the raw data from which radmon generates various displays.

The Data Acquisition consists of two CSUs—File and Net—as shown in Figure 3.2.2-1.

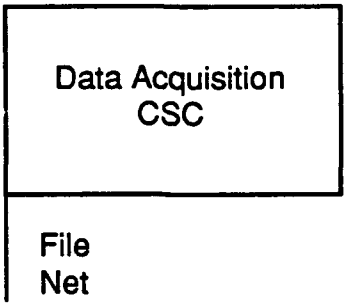


Figure 3.2.2-1. Data Acquisition CSC Structure

3.2.2.1 File CSU

The File CSU consists of routines for reading and writing files.

file_ok

file_ok is the file selection callback. It applies file_io_function to the selected file name. The function call is void file_ok(w, data, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
data	caddr_t	See Appendix.
cback	pointer to XMSelectionBoxCallbackStruct	See Appendix.

file_popup

file_popup pops up a widget on top of other widgets. The function call is void file_popup(w).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.

file_error

file_error pops up an error message for a file io operation. The function call is void file_error(fmt, arg).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fmt	pointer to char	See Appendix.
arg	pointer to char	See Appendix.

Calls	
Function	**Where Described**
file_popup	file.c

file_flush

file_flush brings the display up to date before continuing with the file operation. The function call is void file_flush().

file_inform

file_inform pops up the file progress widget containing the specified message and initializes the scale widget. The function call is void file_inform(fmt, arg, maximum).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fmt	pointer to char	See Appendix.
arg	pointer to char	See Appendix.
maximum	int	See Appendix.

Calls	
Function	**Where Described**
file_flush	file.c
file_popup	file.c

file_inform_end

file_inform_end pops down the file_inform_widget. The function call is void file_inform_end().

file_gets

file_gets does fgets on the input file and count lines and updates the file_inform_widget as reading progresses. The function call is char *file_gets(buf, size, f).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
buf	pointer to char	See Appendix.
size	int	See Appendix.
f	pointer to FILE	See Appendix.

Return Values		
Return Value	**Type**	
NULL	char	
buf	char	

Calls	
Function	**Where Described**
function	

file_count

file_count increments the lines which have been written to the output file and updates the file_inform_widget to indicate progress. The function call is void file_count(n).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
n	int	See Appendix.

file_write_file

file_write_file is a file_io_function to write a radmon file. The selected file name is massaged to add a .radmon extension if necessary, the file is opened and the various parts of the file written. The function call is void file_write_file(filename).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
filename	pointer to char	See Appendix.

Calls	
Function	**Where Described**
file_error(fmt, arg)	file.c
file_inform(fmt, arg, maximum)	file.c
file_inform_end()	file.c
file_count()	file.c
net_count_file(f)	net.c
net_write_file(f)	net.c
DisablePeriodic()	radmon.c
state_count_file()	state.c
state_write_file(f)	state.c
tune_count_file()	tune.c
tune_write_file(f)	tune.c

file_write

file_write activates the callback for the "write" button. It pops up a file selection widget to select a filename, and the file_io_function is set to file_write_file. The function call is void file_write().

Calls	
Function	**Where Described**
file_popup(w)	file.c

file_read_file

file_read_file is a file_io_function to read a radmon file. The selected file is opened and the various parts of the file read. A failure deletes all the information read and operation reverts to on-line mode. The function call is void file_read_file(filename).

filename	pointer to char	See Appendix.

Calls	
Function	**Where Described**
file_error(fmt, arg)	file.c
file_inform(fmt, arg, maximum)	file.c
file_inform_end()	file.c
file_gets(buf, size, f)	file.c
net_read_file(f, version)	net.c
DrainPDUs()	net.c
DisablePeriodic()	radmon.c
DestroyEverything()	radmon.c
Offline()	radmon.c
Online()	radmon.c
state_read_file(f, version)	state.c
tune_read_file(f, version)	tune.c

file_read

file_read activates the callback for the "read" button. It pops up a file selection widget to select a filename, and the file_io_function is set to file_read_file. The function call is void file_read().

Calls	
Function	**Where Described**
file_popup(w)	file.c

fileInitWidgets

fileInitWidgets gets widget IDs for file widgets. The function call is void fileInitWidgets().

Calls	
Function	**Where Described**
NameToWidget(parent, name)	display.c

3.2.1.2 Net CSU

The Net CSU contains routines that manage the interface between the radio monitor and the simulation Ethernet.

InitSimNetwork

InitSimNetwork initializes processing of Ethernet communications. It opens and initializes the network interface, obtains the network address, and subscribes to the simulation, data collection, and radio protocols for the current exercise. The function call is void InitSimNetwork().

Calls	
Function	**Where Described**
htInit(size, key)	hTable.c

net_destroy

net_destroy releases net resources and reset to ground state. The function call is void net_destroy().

Calls	
Function	**Where Described**
htInit(size, key)	hTable.c
htFree(ht)	hTable.c

VehicleIDtoIndex

VehicleIDtoIndex converts a vehicle ID into the net_vehicles index. It looks up the vehicle id in the vehicle id hash table and returns the value found there, if any. If the vehicle is not found, an element of net_vehicles is allocated and entered into the hash table. The function call is int VehicleIDtoIndex(vehicleID).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vehicleID	VehicleID	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
-1	int	No entries available
vidx	int	Vehicle id from vehicle id hash table

Calls	
Function	**Where Described**
htInsert(ht, name, value)	hTable.c
htFind(ht, name, value)	hTable.c

net_read_file

net_read_file reads the net data from a radmon file. The net data consists of the vehicle descriptions and the radio descriptions. The function call is char *net_read_file(f, version).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
f	pointer to FILE	See Appendix.
version	int	See Appendix.

Return Values		
Return Value	**Type**	
"Premature EOF while reading %s"	char	
"File %s has bad format"	char	
"Bad vehicle ID in %s"	char	
"Bad radio ID in %s"	char	
err		
NULL		

Calls	
Function	**Where Described**
file_gets(buf, size, f)	file.c
map_vehicle_appeared(vp1)	map.c
map_init_radio(rp)	map.c
VehicleIDtoIndex(vehicleID)	net.c
netReplenish()	net.c
tlReadFile(f, rp)	timeline.c

net_count_file

net_count_file predicts how many lines of output will be generated when writing the net information portion of a radmon file. The function call is int net_count_file(f).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
f	pointer to FILE	See Appendix.

Return Values		
Return Value	**Type**	
count	int	

Calls	
Function	**Where Described**
tlCountFile(rp)	timeline.c

net_write_file

net_write_file writes the net portion of a radmon file. The function call is void net_write_file(f).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
f	pointer to FILE	See Appendix.

Calls	
Function	**Where Described**
tlWriteFile(f, rp)	timeline.c
file_count(n)	file.c
vehicle_name(vp)	radmon.c

netReplenish

netReplenish replenishes resources to be consumed at "interrupt" level. Interrupt routines are not permitted to malloc memory, so we preallocate memory here. The function call is void netReplenish().

Calls	
Function	**Where Described**
InitVehicle(vp, vidx)	radmon.c

ReadPDUs

ReadPDUs reads and processes PDUs received from the simulation network. The function call is void ReadPDUs().

Calls	
Function	**Where Described**
ProcessVehicleAppearancePDU(pdu, timestamp)	net.c
ProcessDeactivatePDU(pdu, timestamp)	net.c
ProcessStatusChangePDU(pdu, timestamp)	net.c
ProcessVehicleStatusPDU(pdu, timestamp)	net.c
ProcessTransmitterPDU(pdu, timestamp)	net.c
ProcessReceiver PDU(pdu, timestamp)	net.c

DrainPDUs

DrainPDUs reads and discards all pending PDUs and eliminates old PDUs. The function call is void DrainPDUs().

netChangeState

netChangeState records a new state for a radio. The first time a radio is mentioned, its initial state is created and its position in the net_radios vector is established. The function call is void netChangeState(rp, st, timestamp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.
st	STATEP	See Appendix.
timestamp	MSEC	See Appendix.

Calls	
Function	**Where Described**
map_init_radio(rp)	map.c
state_find(st)	state.c
tlChangeState(rp, timestamp)	timeline.c

ProcessVehicleAppearancePDU

ProcessVehicleAppearancePDU marks a vehicle as existing. The first time a vehicle is seen, its position in the net_vehicles vector is established. The function call is void ProcessVehicleAppearancePDU(pdu, timestamp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pdu	pointer to Vehicle Appearance Variant	See Appendix.
timestamp	MSEC	See Appendix.

Calls	
Function	**Where Described**
map_vehicle_appeared(vp1)	map.c
VehicleIDtoIndex(vehicleID)	net.c
netChangeState(rp, st, timestamp)	net.c

ProcessDeactivatePDU

The function call is void ProcessDeactivatePDU(pdu, timestamp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pdu	pointer to Vehicle Appearance Variant	See Appendix.
timestamp	MSEC	See Appendix.

Calls	
Function	**Where Described**
VehicleIDtoIndex(vehicleID)	net.c

ProcessStatusChangePDU

The function call is void ProcessStatusChangePDU(pdu, timestamp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pdu	pointer to Vehicle Appearance Variant	See Appendix.
timestamp	MSEC	See Appendix.

Calls	
Function	**Where Described**
VehicleIDtoIndex(vehicleID)	net.c
netChangeState(rp, st, timestamp)	net.c

ProcessVehicleStatusPDU

The function call is void ProcessVehicleStatusPDU(pdu, timestamp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pdu	pointer to Vehicle Appearance Variant	See Appendix.
timestamp	MSEC	See Appendix.

Calls	
Function	**Where Described**
VehicleIDtoIndex(vehicleID)	net.c
netChangeState(rp, st, timestamp)	net.c

ProcessTransmitterPDU

ProcessTransmitterPDU updates what is known about a transmitter from a Transmitter PDU describing it. The function call is void ProcessTransmitterPDU(pdu, timestamp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pdu	pointer to Vehicle Appearance Variant	See Appendix.
timestamp	MSEC	See Appendix.

Calls	
Function	**Where Described**
VehicleIDtoIndex(vehicleID)	net.c
netChangeState(rp, st, timestamp)	net.c
tune_find(mode, frequency, hopinfo)	tune.c

ProcessReceiverPDU

ProcessReceiverPDU updates what is known about a receiver from a Receiver PDU describing it. The function call is void ProcessReceiverPDU(pdu, timestamp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pdu	pointer to Vehicle Appearance Variant	See Appendix.
timestamp	MSEC	See Appendix.

Calls	
Function	**Where Described**
VehicleIDtoIndex(vehicleID)	net.c
netChangeState(rp, st, timestamp)	net.c

net_geteaddr

net_geteaddr gets a string for our net address. The function call is char *net_geteaddr().

Return Values		
Return Value	**Type**	**Meaning**
print_eaddr	char	Prints network address.

3.2.3 Graphical User Interface CSC

This CSC processes the messages captured by the Data Acquisition CSC and provides a means for displaying information about them. It consists of four second-level CSCs: the Connectivity Display CSC, Status Summary CSC, Strip Chart CSC, and Statistics Summary CSC, as shown in Figure 3.2.3-1.

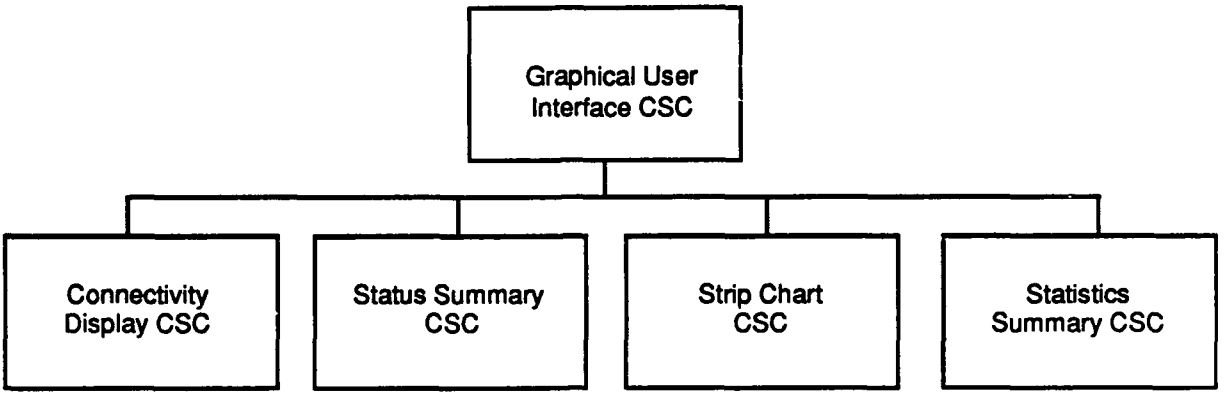


Figure 3.2.3-1. Graphical User Interface CSC Structure

3.2.3.1 Connectivity Display CSC

The Connectivity Display CSC provides map display functions and allows the user to see vehicles with radios and their connectivity. Vehicles are represented by rectangles and have radio rectangles attached to either end; the color of the radio rectangle denotes whether it is receiving, not receiving, transmitting, or doing nothing. Connectivity between pairs of radios is represented by lightning bolts between them; the shape and color of the bolts represent the state of connectivity. The display is updated as radios change states. It can be frozen and stepped from one state change to the next, or stepped to the next anomalous state.

The Connectivity Display CSC is comprised of the CSUs shown in Figure 3.2.3-2.

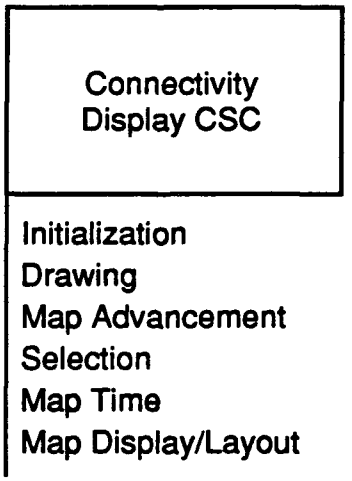


Figure 3.2.3-2. GUI—Connectivity Display CSC Structure

Initialization CSU

The Initialization CSU contains code for initializing various elements of the Connectivity Display such as color and map styles.

ForegroundColorDefault

ForegroundColorDefault is an externally defined routine that returns an XColor. The call is void ForegroundColorDefault (widget, offset, valPtr).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	See Appendix.
offset	int	See Appendix.
valPtr	XRMValuePtr	See Appendix.

BackgroundColorDefault

BackgroundColorDefault is an externally defined routine that returns an XColor. First _XmBackgroundColorDefault is called, and then the resultant Pixel is queried to produce an lor. The call is void BackgroundColorDefault (widget, offset, valPtr).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	See Appendix.
offset	int	See Appendix.
valPtr	XRMValuePtr	See Appendix.

CvtStringToMapStyle

CvtStringToMapStyle is an externally defined routine that converts map style keywords to the corresponding style constant. Its call is void CvtStringToMapStyle (args, nargs, from to).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
args	pointer to XrmValue	See Appendix.
nargs	pointer to int	See Appendix.
from, to	pointer to XrmValue	See Appendix.

mapRegisterNames

This function registers callbacks and other values and adds a converter for MapStyle. The function call is void mapRegisterNames ().

mapInitWidgets

mapInitWidgets gets widget pointers. The function call is void mapInitWidgets ().

Calls	
Function	**Where Described**
NameToWidget(parent, name)	display.c

map_init_radio

This function initializes the map information for a radio. The function call is void map_init_radio (rp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.

Calls	
Function	**Where Described**
state_find(st)	state.c

map_allocate_colors

This function allocates colors for the map display. Only four writeable colors are needed, but all colors are allocated. The function call is void map_allocate_colors ().

map_animation_pixmap

This function creates one of eight pixmaps containing a succession of stripes in four colors which, when animated, will appear to move north, northeast, east, southeast, etc. The function call is Pixmap map_animation_pixmap (dir, rootwindow).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
dir	int	See Appendix.
rootwindow	Window	See Appendix.

Return Values		
Return Value	**Type**	
p	Pixmap	

Drawing CSU

The Drawing CSU contains code for drawing and erasing lightning bolts that show connectivity.

map_bounding_box

This function updates an area that needs to be redrawn by erasing a lightning bolt. The area is simply the bounding rectangle covering all bolts which have been erased.

The function call is void map_bounding_box (pts, npts).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pts	pointer to register XPoint	See Appendix.
npts	int	See Appendix.

map_draw_bolt

This function draws a lightning bolt between two radios according to indicated connectivity types. The shape of the bolt has already been determined by the caller (see map_shape_of_types). The function call is Boolean map_draw_bolt (dpy, wind, rp1, rp2, type1, type2, shape, mode).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
dpy	pointer to Display	See Appendix.
wind	Window	See Appendix.
rp1, rp2	RadioInfoP	See Appendix.
type1, type2	unsigned char	See Appendix.
shape	unsigned char	See Appendix.

mode	DRAW_+MODE	map.c

Return Values		
Return Value	**Type**	**Meaning**
False	Boolean	nothing to do; omit small separations
True	Boolean	

Calls	
Function	**Where Described**
map_bounding_box(pts, npts)	map.c
map_style_of_type(type)	map.c
map_gc_of_type(type, lesser_type, gcp1, gcp2)	map.c

map_reset_pending_draw

This function resets the map_pending_draw information to empty. The function call is void map_reset_pending_draw ().

map_draw

This function draws bolts on the map. This procedure can be interrupted by pending events, so the state is kept in map_pending_draw.

The procedure is as follows: For every pair of selected radios, connectivity types are examined to determine a drawing mode. If the types are marked as new, the mode is draw_mode_force. If there is a non-empty box, the drawing mode is draw_mode_box. Otherwise nothing is drawn. In this last case, the pair is examined further to determine if the bolt requires animation in order to maintain the map_some_animated flag; otherwise this flag is maintained as a side-effect of redrawing the bolt.

If the bolt is to be drawn, its shape is determined and the bolt is drawn using map_draw_bolt. The type and shape are recorded in the connectivity arrays.

After all bolts have been (re)drawn, color-map animation is started if necessary.

The function call is void map_draw ().

Calls	
Function	**Where Described**
map_style_of_type(type)	map.c
map_shape_of_types(type1, type2)	map.c
map_draw_bolt(dpy, wind, tp1, rp2, type1, type2, shape, mode)	map.c

map_start_draw

This function resets map_pending_draw state and starts drawing. The function call is void map_start_draw().

Calls	
Function	**Where Described**
map_draw()	map.c

map_erase

This routine is similar to map_draw except that bolts are erased according to their current shape instead of being drawn in their new shape. Erasure occurs in anticipation of a subsequent draw. Thus if the draw operation will completely cover the current bolt, no erasure is necessary. This is an important optimization because the shape being erased is frequently covered by the shape being drawn, and if the erasure is done it forces redrawing of all the bolts which intersect the erased bolt.

As bolts are erased, the area erased is accumulated to determine which bolts will need to be redrawn.

The rules about which shapes cover which other shapes are:

1. Every shape covers itself.
2. Everything covers the invisible shape.
3. The invisible shape covers nothing but itself.
4. The blunt_shape and sharp_blunt shapes cover the sharp_sharp and blunt_blunt shapes.
5. The sharp_sharp and blunt_blunt shapes cover each other.

The function call is void map_erase ().

Calls	
Function	**Where Described**
map_style_of_type(type)	map.c
map_shape_of_types(type1, type2)	map.c
map_draw_bolt(dpy, wind, rep1, rp2, type1, type2, shape, mode)	map.c
map_start_draw()	map.c

map_start_erase

This function resets map_pending_draw and starts erasing. The function call is map_start_erase ().

Calls	
Function	**Where Described**
map_erase()	map.c

map_expose

If a map_draw operation is in progress, it is cancelled. If a map_erase operation is in progress, it is left alone. Then the exposed area is added to the map_pending_draw area. If a map_erase operation is not in progress, a map_draw is started.

The function call is void map_expose (w, a, callback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
a	caddr_t	See Appendix.
back	pointer to XMDrawingArea CallbackStruct	See Appendix.

Calls	
Function	**Where Described**
map_start_draw()	map.c

map_style_of_type

This routine converts a connectivity type to a lightning bolt style. The default bolt shape is unconnected_style. The function call is MapStyle map_style_of_type (type).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	unsigned char	See Appendix.

Return Values		
Return Value	**Type**	
untuned_style	MapStyle	
tuned_style	MapStyle	
unconnected_style	MapStyle	
connected_style	MapStyle	
conflict_style	MapStyle	
receiving_style	MapStyle	

map_shape_of_types

This function returns the appropriate bolt shape for the connectivity types between a pair of radios. The function call is unsigned char map_shape_of_types (type1, type2).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type1	unsigned char	See Appendix.
type2	unsigned char	See Appendix.

Return Values		
Return Value	**Type**	
MAP_SHAPE_SHARP_ SHARP	unsigned char	
MAP_SHAPE_SHARP_ BLUNT	unsigned char	
MAP_SHAPE_BLUNT_ SHARP	unsigned char	
MAP_SHAPE_BLUNT_ BLUNT	unsigned char	

map_gc_of_type

This function converts the connectivity types between two radios to the graphics contexts for filling the two halves of a lightning bolt. The function call is void map_gc_of_type (type, lesser_type, gcp1, gcp2).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	unsigned char	See Appendix.
lesser_type	unsigned char	See Appendix.
gcp1, gcp2	pointer to GC	See Appendix.

map_set_connectivity

This function sets the connectivity of "rp1" from the "idx" transmitter to "new_connectivity". It filters out redundant connectivity changes and marks the connectivity as "con_new". It then returns bits indicating what kind of changes occurred. The function call is int map_set_connectivity (rp1, idx, new_connectivity).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp1	RadioInfoP	See Appendix.
idx	int	See Appendix.
new_connectivity	unsigned char	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
4	int	MAP_BECAME_DIFFERENT
2	int	MAP_BECAME_NOT_ ANOMALOUS
1	int	MAP_BECAME_ ANOMALOUS
0	int	

map_color_animate

This is an interval timer callback for doing color-map animation. It switches the colors for the animated lightning bolts to the next phase. The timer is restarted if there are still some animated bolts present; otherwise, the animation stops. The function call is void map_color_animate ().

Map Advancement CSU

map_input

This function is the input callback for vehicle boxes. Input processing is only performed to permit a vehicle box to be dragged to a new location. The function call is void map_input (w, vp, event).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
vp	VehicleInfoP	See Appendix.
event	pointer to XEvent	See Appendix.

Calls	
Function	**Where Described**
map_color_radio(rp)	map.c
map_color_vehicle(vp)	map.c
map_display_move(visible, x, y)	map.c

map_step

This is the callback function for activating the step button. It calls map_advance to advance one step. The function call is void map_step ().

Calls	
Function	**Where Described**
map_advance(up_to, advance_mode, max_advance)	map.c
traffic_set_cursor_time(time)	traffic.c

map_step_to_anomaly

This callback function activates the anomaly button. It calls map_advance to advance to the next anomaly. The function call is void map_step_to_anomaly ().

Calls	
Function	**Where Described**
map_advance(up_to, advance_mode, max_advance)	map.c
traffic_set_cursor_time(time)	traffic.c

map_freeze

This is the callback function for activating the freeze button. It desensitizes the freeze button and sensitizes the unfreeze, step, and anomaly buttons. It also allows input to the time widget. If the traffic display is already frozen and has a valid cursor, the map time is advanced to the traffic cursor time. Otherwise the traffic cursor time is set to the map time. The function call is void map_freeze ().

Calls	
Function	**Where Described**
map_advance(up_to, advance_mode, max_advance)	map.c
traffic_set_cursor_time(time)	traffic.c
traffic_cursor_time(p)	traffic.c

map_unfreeze

This is the callback function for activating the unfreeze button. It desensitizes the step, anomaly, and unfreeze widgets, sensitizes the freeze button, and disables input to the time widget. The map time is advanced to the current time. The function call is void map_unfreeze ().

Calls	
Function	**Where Described**
map_advance(up_to, advance_mode, max_advance)	map.c

map_reset

This function resets the map. It is called when a new set of data is started (either read from a file or when the user restarts on-line collection). The function call is void map_reset ().

Calls	
Function	**Where Described**
map_unfreeze()	map.c
map_set_time(t)	map.c
map_init_radio(rp)	map.c
map_freeze()	map.c

map_radio_transmit_time

This function finds the latest time before the indicated time when the radio transmitted. The function call is MSEC map_radio_transmit_time (rp, when).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.
when	MSEC	See Appendix.

Return Values		
Return Value	**Type**	
0	MSEC	
tle->tle_when	MSEC	

Calls	
Function	**Where Described**
tlPrev(tlsp, tlep)	timeline.c
tlFind(rp, when, tlsp, tlep)	timeline.c

map_set_radio_time

map_set_radio_time sets the pointer into a radio's data to the location corresponding to the specified time. The function call is void map_set_radio_time (rp, when).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.
when	MSEC	See Appendix.

Calls	
Function	**Where Described**
state_find(st)	state.c
tlFind(rp, when, tlsp, tlep)	timeline.c

map_advance

map_advance advances the map display one step. First it finds which radio has the earliest next state. If this state has a new tuning, it changes all connectivity to either con_tuned or con_untuned, depending on whether the other radio is tuned the same or not.

Next we consider whether the state transition is the initiation or termination of transmission or reception. In each case, we update the connectivity accordingly. A complication is necessary due to the possibility that PDU transmission delays allow cause and effect to be reversed; the beginning of transmission may follow the beginning of reception. This is resolved by examining the following state of affected radios to avoid false indications of lack of connectivity.

Start of Transmission: Scan all receivers tuned to this frequency. A receiver which is tuned the same as the transmitter should be receiving from that transmitter or should be doing so in the next state. If it is not, then there is either a conflict or a lack of connectivity. A conflict is indicated when the receiver is busy receiving from a different transmitter or is itself transmitting. These two connectivity changes are made now; otherwise, we wait until the next state when the onset of reception will be recorded.

Start of Reception: When a receiver begins receiving from a transmitter, the connectivity becomes con_receiving.

End of Transmission: Nothing is done on this transition. All the work is done at the end of reception at the receivers.

End of Reception: Reception can cease due to four causes: end of transmission, loss of signal, reception of a stronger signal (SC mode only), or start of transmission by the receiver (double push-to-talk). End of transmission is determined by examining the state of the transmitter (or its next state) to see if it is still transmitting on the same frequency. If not, then this is a normal end of reception and the connectivity is marked con_connected. If the transmitter is still transmitting, then the next state of the receiver is examined. If that state is only a short time in the future and is ST_TRANSMITTING or ST_RECEIVING, then the connectivity becomes con_conflict. Otherwise, the connectivity becomes con_unconnected

The function call is void map_advance (up_to, advance_mode, max_advance).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
up_to	MSEC	See Appendix.
advance_mode	MAP_ADVANCE_MODE	map.c
max_advance	MSEC	See Appendix.

Calls	
Function	**Where Described**
map_set_radio_time(rp, when)	map.c
map_display()	map.c
map_set_time(t)	map.c
map_update()	map.c
map_set_connectivity(rp1, idx, new_connectivity)	map.c
map_radio_transmit_time(rp, when)	map.c
map_more_radios()	map.c

map_advance_display

This function provides the public interface to map_advance. The function call is void map_advance_display (up_to, max_advance, if_frozen).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
up_to	MSEC	See Appendix.
max_advance	MSEC	See Appendix.
if_frozen	Boolean	See Appendix.

Calls	
Function	**Where Described**
map_display()	map.c
map_advance(up_to, advance_mode, max_advance)	map.c

map_vehicle_vanished

map_vehicle_vanished marks the vehicle as needing recoloring. The function call is void map_vehicle_vanished (vp1).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vp1	VehicleInfoP	See Appendix.

Selection CSU

map_select

This is the callback function for selecting radios to show on the connectivity map. The principal function is to set the r_map.selected boolean for every radio which is selected and to clear the boolean for every radio not selected. The manual placement of the vehicle of newly selected radios is cancelled and vehicle recoloring is scheduled as necessary. map_more_radios is called to schedule the map_display.

The function call is void map_select (w, a, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
a	caddr_t	See Appendix.
cback	pointer to XMListCallbackStruct	See Appendix.

Calls	
Function	**Where Described**
map_more_radios()	map.c

Map Time CSU

map_time_callback

This is the callback function for typing in a new map time. The text of the time widget is parsed into a new time and the map display advanced to that time. If the time cannot be parsed, the time is reset to the current map time. The function call is void map_time_callback (w, a, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
a	caddr_t	See Appendix.
cback	pointer to XMListCallbackStruct	See Appendix.

Calls	
Function	**Where Described**
map_advance(up_to, advance_mode, max_advance)	map.c
map_set_time(t)	map.c
traffic_set_cursor_time	traffic.c

map_set_time

map_set_time changes the value displayed in the map_time_widget. The function call is void map_set_time (t).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
t	MSEC	See Appendix.

Map Display/Layout CSU

map_close

The map_widget is removed from the screen (unmanaged). The function call is void map_close ().

map_display

This function displays the map. First the map_widget is created, if necessary. Then new radio selection buttons are created, if necessary. Then new vehicle widgets are created, if necessary. Then vehicle widgets are remanaged as necessary. The map is remeasured and re-layed out. Then the map_widget is managed if it is not already on the screen; otherwise, it is redrawn if any of the preceding operations require it. The map is forced into the frozen state in offline mode. The function call is void map_display ().

Calls	
Function	**Where Described**
map_freeze()	map.c
map_create_widget()	map.c
map_create_vehicle_widget(vp)	map.c
map_remanage_vehicle(vp)	map.c
map_measure()	map.c
map_layout()	map.c
map_create_radio_buttons()	map.c
map_recolor()	map.c

map_display_move

This function updates the position of a vehicle outline box while dragging the vehicle around. The function call is void map_display_move (visible, x, y).

Parameters		
Parameter	Type	Where Typedef Declared
visible	Boolean	See Appendix.
x, y	Position	See Appendix.

map_update

This function updates the map display to reflect its new state. It recolors everything and starts erasing bolts. The function call is void map_update ().

Calls	
Function	Where Described
map_recolor()	map.c
map_reset_pending_draw()	map.c
map_start_erase()	map.c

map_vehicle_moved

This function schedules a map_display operation for ten seconds from now, if there is not already a map_display pending. The function call is void map_vehicle_moved (vp).

Parameters		
Parameter	Type	Where Typedef Declared
vp	VehicleInfoP	See Appendix.

map_more_radios

This function schedules a map_display operation for 50 milliseconds from now if there is not already one scheduled. If a later map_display is scheduled, it is cancelled. The function call is void map_more_radios ().

map_destroy

map_destroy destroys the radio selection buttons and removes the map display from the screen. The function call is void map_destroy ().

map_create_widget

This function creates the map widget and initializes things. The function call is void map_create_widget ().

Calls	
Function	**Where Described**
NameToWidget(parent, name)	display.c
map_resize(w)	map.c
map_unfreeze()	map.c
map_allocate_colors()	map.c
map_animation_pixmap(dir, rootwindow)	map.c
map_set_time(t)	map.c

map_create_radio_buttons

This function creates selection buttons for radios which have recently appeared. The function call is void map_create_radio_buttons ().

Calls	
Function	**Where Described**
radio_name(rp)	radmon.c

map_create_vehicle_widget

This function creates the widget for a vehicle. The various widget ids are recorded in the vehicle or radio structures. Event handlers are added for dragging the widgets around. The radios are initially unselected. The function call is void map_create_vehicle_widget (vp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vp	VehicleInfoP	See Appendix.

Calls	
Function	**Where Described**
set_widget_label(va_alist)	status.c
NameToWidget(parent, name)	display.c
vehicle_name(vp)	radmon.c

map_remanage_vehicle

This function changes a vehicle's appearance according to the new selection state for the vehicle's radios. The vehicle is unmanaged if neither radio is selected. If a radio is selected, then the label and separator for that radio are managed, if necessary. If a radio is not selected, then the label and separator for that radio are unmanaged, if necessary. If either radio is selected, then the vehicle is managed, if necessary.

The function call is Boolean map_remanage_vehicle (vp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vp	VehicleInfoP	See Appendix.

Return Values		
Return Value	**Type**	
need_redisplay	Boolean	

map_layout_compare_x

This function allows qsort to compare the x coordinate of two vehicles. Ties are broken by the location in memory of the vehicle data. The function call is int map_layout_compare_x (vpp1, vpp2).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vpp1, vpp2	pointer to VehicleInfoP	See Appendix.

Return Values		
Return Value	**Type**	
-1	int	
1	int	
0	int	

map_layout_compare_y

This function allows qsort to compare the y coordinate of two vehicles. Ties are broken by the location in memory of the vehicle data. The function call is int map_layout_compare_y (vpp1, vpp2).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vpp1, vpp2	pointer to VehicleInfoP	See Appendix.

Return Values		
Return Value	Type	
-1	int	
1	int	
0	int	

map_measure

map_measure updates the left and right extension and height measurements for all selected vehicles. The function call is void map_measure ().

map_layout

map_layout lays out the vehicle widgets on the map. Vehicles are placed on a grid such that vehicles are ordered according to their east-west and north-south positions. The function call is Boolean map_layout ().

Return Values		
Return Value	Type	Meaning
False	Boolean	No vehicles to lay out
result	Boolean	True; vehicles to lay out

map_resize

This function records the new dimensions of the map_body_widget. The function call is void map_resize (w).

Parameters		
Parameter	Type	Where Typedef Declared
w	Widget	See Appendix.

map_color_radio

This function changes the background color of the radio label widget according to the radio's state. The function call is void map_color_radio (rp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.

map_color_vehicle

This function changes the background color of the vehicle label widget according to the vehicle's state. The function call is void map_color_vehicle (vp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vp	VehicleInfoP	See Appendix.

map_recolor

This function recolors all vehicles and radios as needed. The function call is void map_recolor ().

Calls	
Function	**Where Described**
map_color_radio(rp)	map.c
map_color_vehicle(vp)	map.c

3.2.3.2 Status Summary (status.c)

The Status Summary CSC includes routines to manipulate the Status Display, which describes the state of selected radios. Selecting Status displays a set of buttons representing all selected radios. Selecting a radio's button displays a box that describes the state of the radio; this state information is updated whenever the state changes. If the traffic display is frozen, the status boxes display the state of the radios at the time selected by the traffic mark. When information in a box changes, the changed information is displayed in blue. Old information is shown in black.

This CSC consists of three CSUs, shown in Figure 3.2.3.2-1.

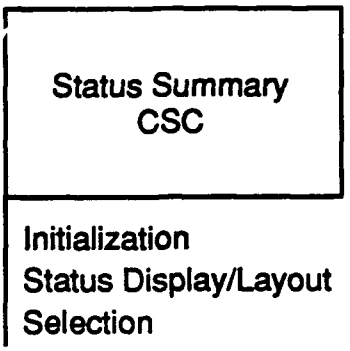


Figure 3.2.3.2-1. GUI—Status Summary CSC Structure

Initialization CSU

The Initialization CSU contains code for initializing the Status Display. Individual functions in this CSU are described below.

status_close_callback

status_close_callback activates the callback for the close button in the status widget and removes the status box from the screen. The function call is void status_close_callback(w, rp, b).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
rp	RadioInfoP	See Appendix.
b	caddr_t	See Appendix.

statusRegisterNames

This function registers names for Mrm. The function call is void statusRegisterNames().

statusInitWidgets

This function gets widget IDs for status widgets. The function call is void statusInitWidgets().

create_status_widget

This function is used for creating a new status widget. The function call is void create_status_widget(rp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.

set_widget_label

This function is essentially SPRINTF for label widgets. The function call is void set_widget_label (va_alist).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
va_alist		See Appendix.

set_status_label

set_status_label is the same as set_widget_label except the foreground color is changed to the highlighted color. The function call is void set_status_label (va_alist).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
va_alist		See Appendix.

Status Display/Layout CSU

This CSU contains code for changing the layout .

getUpdateFlags

getUpdateFlags computes a mask of status fields which differ between two states. The function call is int getUpdateFlags(old_state, new_state).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
old_state	registerSTATEP	See Appendix.
new_state	registerSTATEP	See Appendix.

Return Values		
Return Value	Type	
change	int	

UTMString

UTMString gets the coordinate string for a given position. The function call is char *UTMString (x, y).

Parameters		
Parameter	Type	Where Typedef Declared
x	double	See Appendix.
y	double	See Appendix.

Return Values		
Return Value	Type	Meaning
str	char	Returns static storage

status_new_radio

status_new_radio is called when a new radio appears in order to create its selection button. The function call is void status_new_radio(rp).

Parameters		
Parameter	Type	Where Typedef Declared
rp	RadioInfoP	See Appendix.

status_update

status_update updates the contents of a status box according to the new state of a radio. If the traffic widget has a valid cursor, the time of that cursor is used to find the state to be displayed. In this case, the time is displayed as "Now" to avoid extra repainting. If the traffic cursor is not valid, the status display is update to reflect the current state of the radio. Only changed fields are drawn. The function call is void status_update(rp).

Parameters		
Parameter	Type	Where Typedef Declared
rp	RadioInfoP	See Appendix.

UpdateRadioStatus

UpdateRadioStatus interfaces to status_new_radio and status_update to create all new radio selection buttons and update all radio status displays. The function call is void UpdateRadioStatus().

create_status_select_button

create_status_select_button adds a button to select the status of a new radio. The function call is void create_status_select_button(rp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.

status_destroy

status_destroy deletes all status buttons and status widgets. The function call is void status_destroy().

unhilite_widget

unhilite_widget restores the foreground color of a widget to the unhighlighted color. The function call is void unhilite_widget(w).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.

Selection CSU

This CSU contains code for showing which radios are currently selected.

status_select

status_select is called when the set of selected radios changes. It changes the set of managed status boxes to reflect the selected radios. The function call is void status_select(w, a, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
a	caddr_t	See Appendix.
cback	pointer to XmButtonLCallbackstruct	See Appendix.

3.2.3.3 Stripchart CSC (traffic.c)

The Stripchart CSC includes routines that control the traffic display, which consists of horizontal lines of data that represent the state of radios as a function of time. There are four states: non-existent/inoperative/not-receiving, inactive, transmitting, and receiving. The non-existent/inoperative/not-receiving state is represented by a line the same color as the background (which is therefore invisible). The inactive state is represented by a thin black line, the transmitting state by a thick green line. The receiving state is represented by a thick magenta line.

The Stripchart CSC is comprised of the CSUs shown in Figure 3.2.3.3-1.

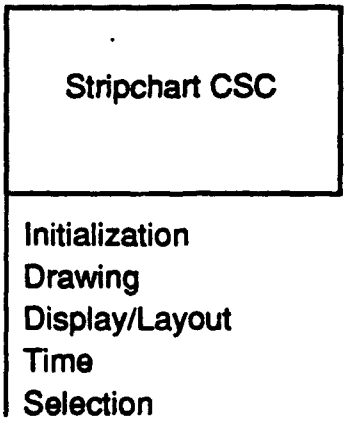


Figure 3.2.3.3-1. GUI—Stripchart CSC Structure

Initialization CSU

The Initialization CSU contains code for initializing the Traffic Display.

trafficRegisterNames

trafficRegisterNames registers names to be used by mrm. The function call is void trafficRegisterNames().

trafficInitWidgets

trafficInitWidgets gets widget ids for the various button lists. The function call is void trafficInitWidgets().

Calls	
Function	**Where Described**
NameToWidget(parent, name)	display.c

traffic_allocate_colors

traffic_allocate_colors allocates colors for the various states represented in the traffic display. It allocates one plane for the cursor so it can be drawn independently of the traffic lines. The function call is void traffic_allocate_colors(trf).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.

Drawing CSU

The Drawing CSU consists of routines for drawing and erasing traffic slots and labels.

traffic_draw_cursor

This function draws the cursor and mark. The function call is void traffic_draw_cursor(trf).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.

traffic_erase_cursor

traffic_erase_cursor erases the cursor and mark. The function call is void traffic_erase_cursor(trf).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.

traffic_erase_labels

traffic_erase_labels erases (and then redraws) the labels for the given slots. The function call is void traffic_erase_labels(trf, first_slot, last_slot).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
first_slot	int	See Appendix.
last_slot	int	See Appendix.

Calls	
Function	**Where Described**
traffic_set_scrollbar(trf)	traffic.c

traffic_erase_drawing

traffic_erase_drawing erases (and then redraws) the traffic drawing. The function call is void traffic_erase_drawing(trf).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.

Calls	
Function	**Where Described**
traffic_set_scrollbar(trf)	traffic.c

traffic_draw_label

traffic_draw_label draws the label for the given slot. The function call is void traffic_draw_label(trf, slot).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
slot	int	See Appendix.

traffic_draw_slot

traffic_draw_slot draws the data for the given slot starting a position "left" on the screen and continuing for "width" pixels. For each state change, a graphics context is selected according to the previous state, and a line is drawn using that graphics context from the previous position to the position of the state change. Transmissions are emphasized by drawing the lines for transmission one pixel farther than for other states. The function call is int traffic_draw_slot(trf, slot, left, width).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
slot	int	See Appendix.
left	int	See Appendix.
width	int	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
right	int	All done

Calls	
Function	**Where Described**
tlFind(rp, when, tlsp, tlep)	timeline.c
TIME_TO_WINDOW_COORD(trf, t)	traffic.c
WINDOW_TO_TIME_COORD(trf, x)	traffic.c
tune_selected(bi, tune)	tune.c

traffic_draw_labels

traffic_draw_labels draws the labels for all slots. The function call is void traffic_draw_labels(trf, first_slot, last_slot).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
first_slot	int	See Appendix.
last_slot	int	See Appendix.

Calls	
Function	**Where Described**
traffic_draw_label	traffic.c

traffic_draw_grid

traffic_draw_grid draws and labels the traffic label. The function call is void traff_draw_grid (trf, left, width).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
left	int	See Appendix.
width		See Appendix.

Calls	
Function	**Where Described**
traffic_format_time(trf, time)	traffic.c
TIME_TO_WINDOW_COORD(trf, t)	traffic.c
WINDOW_TO_TIME_COORD(trf, x)	traffic.c

traffic_draw_slots

traffic_draw_slots draws slots in the indicated region. The function call is void traffic_draw_slots(trf, first_slot, last_slot, left, width).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
first_slot, last_slot	int	See Appendix.
left	int	See Appendix.
width	int	See Appendix.

Calls	
Function	**Where Described**
traffic_draw_grid(trf, left, width)	traffic.c
traffic_draw_slot(trf, slot, left, width)	traffic.c

traffic_redraw_labels

traffic_redraw_labels exposes the callback for the label area and redraws the labels included in the exposed area. The function call is void traffic_redraw_labels(w, trf, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
cback	pointer to XmDrawingAreaCallbackStruct	See Appendix.

Calls	
Function	**Where Described**
traffic_draw_labels(trf, first_slot, last_slot)	traffic.c

traffic_redraw

traffic_redraw redraws traffic slots. The function call is void traffic_redraw(trf, left, top, width, height).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
left	int	See Appendix.
top	int	See Appendix.
width	int	See Appendix.
height	int	See Appendix.

Calls	
Function	**Where Described**
traffic_draw_slots(trf, first_slot, last_slot, left, width)	traffic.c

traffic_expose

traffic_expose exposes the callback for the traffic display and redraws the exposed slots. The function call is void traffic_expose(w, trf, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
cback	pointer to XmDrawingAreaCallbackStruct	See Appendix.

Calls	
Function	**Where Described**
traffic_redraw(trf, left, top, width, height)	traffic.c

traffic_graphics_expose

traffic_graphics_expose provides the graphics expose callback for the traffic display and redraws the exposed slots. The function call is void traffic_graphics_expose(w, trf, event).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
event	pointer to XEvent	See Appendix.

Calls	
Function	**Where Described**
traffic_redraw(trf, left, tp, width, height)	traffic.c

traffic_resize

This function is the resize callback for traffic drawing widget. It records the new dimensions, sets the scrollbar parameters, and recomputes the grid parameters. The concomitant redrawing is initiated by the expose event. The function call is void traffic_resize(w, trf, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
cback	pointer to XmDrawingAreaCallbackStruct	See Appendix.

Calls	
Function	**Where Described**
traffic_set_scrollbar(trf)	traffic.c
traffic_measure_grid(trf)	traffic.c

traffic_set_offset

traffic_set_offset changes the drawing offset to a new value. It calls traffic_left or traffic_right as needed to accomplish this. The function call is traffic_set_offset(trf, new_offset).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
new_offset	int	See Appendix.

Calls	
Function	**Where Described**
traffic_left(trf, delta, limit)	traffic.c
traffic_right(trf, delta)	traffic.c

traffic_new_height

traffic_new_height computes a new height for the traffic widget and changes it if necessary. The function call is void traffic_new_height(trf).

Parameters		
Parameter	Type	Where Typedef Declared
trf	TRAFFICINFOP	See Appendix.

Display/Layout CSU

The Display/Layout CSU contains code for laying out and displaying items on the traffic display.

TIME_TO_WINDOW_COORD

This function converts from time to position in the traffic window. The function call is int TIME_TO_WINDOW_COORD(trf,t).

Parameters		
Parameter	Type	Where Typedef Declared
trf	TRAFFICINFOP	See Appendix.
t	MSEC	See Appendix.

Return Values		
Return Value	Type	
2147483647	int	
-2147483647	int	
x	int	

WINDOW_TO_TIME_COORD

This function converts from window position to time. The function call is MSEC WINDOW_TO_TIME_COORD(trf,x).

Parameters		
Parameter	Type	Where Typedef Declared
trf	TRAFFICINFOP	See Appendix.
x	int	See Appendix.

Return Values		
Return Value	Type	
2147483647	int	
-2147483647	int	

t	MSEC	

TIME_TO_WINDOW_DELTA

This function converts from a time dimension to a window dimension. The function call is int TIME_TO_WINDOW_DELTA(trf,t).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
t	MSEC	See Appendix.

Return Values		
Return Value	**Type**	
2147483647	int	
-2147483647	int	
x	int	

WINDOW_TO_TIME_DELTA

This function converts from a window dimension to a time dimension. The function call is MSEC WINDOW_TO_TIME_DELTA(trf,x).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
x	int	See Appendix.

Return Values		
Return Value	**Type**	
2147483647	int	
-2147483647	int	
t	MSEC	

traffic_set_scrollbar

traffic_set_scrollbar updates scrollbar variables to reflect a new position or size. The function call is void traffic_set_scrollbar(trf).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.

Calls	
Function	**Where Described**
WINDOW_TO_TIME_COORD(trf, x)	traffic.c
WINDOW_TO_TIME_DELTA(trf, x)	traffic.c

traffic_measure_grid

traffic_measure_grid figures out how to lay out and label the grid on the traffic display. The grid is constrained to increment by the listed amounts and the grid interval is selected to place no more than 8 ticks on the display. If the number of ticks chosen would result in labels overlapping, the number of ticks is reduced. The function call is Boolean traffic_measure_grid(trf).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.

Return Values		
Return Value	**Type**	
orig_grid_first_format != trf->tr_grid_first_format |orig_grid_last_format != trf->tr_grid_last_format || orig_grid_scale != trf->tr_grid_scale || orig_grid_label_width != trf->tr_grid_label_width	Boolean	

Calls	
Function	**Where Described**
WINDOW_TO_TIME_COORD(trf, x)	traffic.c
TIME_TO_WINDOW_DELTA(trf, t)	traffic.c
traffic_format_time(trf, time)	traffic.c
traffic_time_to_date_field(time)	traffic.c

traffic_close

This function activates the callback for the "close" button and removes the traffic widget from the screen. The function call is void traffic_close(w, trf, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
cback	caddr_t	See Appendix.

traffic_zoom_in

traffic_zoom_in activates the callback for the zoom in button. It adjusts the scale and offset of the drawing so the area between the mark and cursor falls between the left and right extremes of the display. If the cursor is not valid, the scale is doublesd, keeping the left edge constant. The function call is void traffic_zoom_in(w, trf, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
cback	caddr_t	See Appendix.

Calls	
Function	**Where Described**
WINDOW_TO_TIME_COORD(trf, x)	traffic.c
TIME_TO_WINDOW_DELTA(trf, t)	traffic.c
traffic_cursor_invalid(trf)	traffic.c
traffic_set_scrollbar(trf)	traffic.c
traffic_erase_drawing(trf)	traffic.c
traffic_measure_grid(trf)	traffic.c
traffic_freeze(w, trf, cback)	traffic.c

traffic_zoom_out

traffic_zoom_out activates the callback for the zoom out button. It adjusts the scale and offset of the drawing so the area between the left and right extremes of the display falls between the mark and cursor. If the cursor is not valid, the scale is halved, keeping the left edge constant. The function call is void traffic_zoom_out(w, trf, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
cback	caddr_t	See Appendix.

Calls	
Function	**Where Described**
TIME_TO_WINDOW_COORD(trf, t)	traffic.c
WINDOW_TO_TIME_COORD(trf, x)	traffic.c
TIME_TO_WINDOW_DELTA(trf, t)	traffic.c
traffic_cursor_invalid(trf)	traffic.c
traffic_set_scrollbar(trf)	traffic.c
traffic_erase_drawing(trf)	traffic.c
traffic_measure_grid(trf)	traffic.c
traffic_freeze(w, trf, cback)	traffic.c

traffic_left

traffic_left scrolls the traffic display left by the indicated amount. The scroll distance is limited so that the right data extreme is not less than "limit". The cursor is moved so it stays with the data. The grid is remeasured. The function returns False if no scrolling is possible. The function call is Boolean traffic_left(trf, delta, limit).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
delta	int	See Appendix.
limit	int	See Appendix.

Return Values		
Return Value	**Type**	
result	Boolean	

Calls	
Function	**Where Described**
TIME_TO_WINDOW_COORD(trf, t)	traffic.c
traffic_draw_cursor(trf)	traffic.c
traffic_erase_cursor(trf)	traffic.c
traffic_erase_drawing(trf)	traffic.c
traffic_measure_grid(trf)	traffic.c
traffic_draw_slots(trf, first_slot, last_slot, left, width)	traffic.c

traffic_continue_pan_left

This function is a timer routine to perform another step of panning to the left. It restarts another timer if scrolling occurs. The function call is void traffic_continue_pan_left(trf, id).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
id	XtIntervalId	See Appendix.

Calls	
Function	**Where Described**
traffic_left	traffic.c
traffic_set_scrollbar(trf)	traffic.c

traffic_pan_left

traffic_pan_left initiates or terminates panning left. Arming the pan left button starts panning. Disarming terminates panning. The function call is void traffic_pan_left(w, trf, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
cback	caddr_t	See Appendix.

Calls	
Function	**Where Described**
traffic_continue_pan_left	traffic.c

traffic_right

traffic_right scrolls the traffic display right by the indicated amount. The scroll distance is limited so that the left data extreme is not greater than 10. The cursor is moved so it stays with the data. The grid is remeasured. The function returns False if no scrolling is possible. The function call is Boolean traffic_right(trf, delta).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
delta	int	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
False	Boolean	No more panning
delta==original_delta	Boolean	

Calls	
Function	**Where Described**
traffic_draw_cursor(trf)	traffic.c
traffic_erase_cursor(trf)	traffic.c
traffic_erase_drawing(trf)	traffic.c
traffic_measure_grid(trf)	traffic.c
traffic_draw_slots(trf, first_slot, last_slot, left, width)	traffic.c
traffic_freeze(w, trf, cback)	traffic.c

traffic_continue_pan_right

traffic_continue_pan_right is a timer routine to perform another step of panning to the right. It restarts another timer if scrolling occurs. The function call is void traffic_continue_pan_right(trf, id).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
id	XtIntervalId	See Appendix.

Calls	
Function	**Where Described**
traffic_set_scrollbar(trf)	traffic.c
traffic_right(trf, delta)	traffic.c

traffic_pan_right

traffic_pan_right initiates or terminates panning right. Arming the pan right button starts panning; disarming terminates panning. The function call is void traffic_pan_right(w, trf, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
cback	caddr_t	See Appendix.

Calls	
Function	**Where Described**
traffic_continue_pan_right(trf, id)	traffic.c

traffic_fit

traffic_fit is the callback for the fit button. It adjusts the scale and offsets to fit the entire data in the window. The function call is void traffic_fit(w, trf, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
cback	caddr_t	See Appendix.

Calls	
Function	**Where Described**
traffic_unfreeze(w, trf, cback)	traffic.c
traffic_set_scrollbar(trf)	traffic.c
traffic_erase_drawing(trf)	traffic.c
traffic_measure_grid(trf)	traffic.c

traffic_freeze

traffic_freeze is the callback for freeze button. It prevents the traffic display from scrolling, sensitizes the unfreeze button, and desensitizes the freeze button. The function call is void traffic_freeze(w, trf, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
cback	caddr_t	See Appendix.

traffic_unfreeze

traffic_unfreeze is the callback for unfreeze button. It allows the traffic display to scroll, sensitizes the freeze button, and desensitizes the unfreeze button. The function call is void traffic_unfreeze(w, trf, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
cback	caddr_t	See Appendix.

Calls	
Function	**Where Described**
traffic_cursor_invalid(trf)	traffic.c
traffic_erase_cursor(trf)	traffic.c

traffic_input

traffic_input is the input callback for the traffic drawing widget. It sets the cursor and mark to where the left button is pressed and released respectively. The function call is PRIVATE void traffic_input(w, trf, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
cback	caddr_t	See Appendix.

Calls	
Function	**Where Described**
map_advance_display(up_to, max_advance, if_frozen)	map.c
WINDOW_TO_TIME_COORD(trf, x)	traffic.c
traffic_cursor_valid(trf)	traffic.c
traffic_draw_cursor(trf)	traffic.c
traffic_erase_cursor(trf)	traffic.c
traffic_freeze(w, trf, cback)	traffic.c

traffic_scrollbar

traffic_scrollbar is the callback for the traffic scrollbar. It cCalls traffic_set_offset to set the new position as selected by the scrollbar. The function call is void traffic_scrollbar(w, trf, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
cback	caddr_t	See Appendix.

Calls	
Function	**Where Described**
traffic_set_offset(trf, new_offset)	traffic.c
WINDOW_TO_TIME_COORD(trf, x)	traffic.c
TIME_TO_WINDOW_COORD(trf, t)	traffic.c
traffic_set_scrollbar(trf)	traffic.c

traffic_motion

traffic_motion updates the mark position as the mouse is dragged across the traffic display. The function call is void traffic_motion(w, trf, event).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
trf	TRAFFICINFOP	See Appendix.
event	pointer to XEvent	See Appendix.

Calls	
Function	**Where Described**
traffic_draw_cursor(trf)	traffic.c
traffic_erase_cursor(trf)	traffic.c

traffic_create_widget

traffic_create_widget creates a traffic widget and fills in its associated data structure. The function call is void traffic_create_widget(trf).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.

Calls	
Function	**Where Described**
traffic_measure_grid(trf)	traffic.c
traffic_allocate_colors(trf)	traffic.c
NameToWidget(parent, name)	display.c
set_widget_label(va_alist)	status.c
traffic_cursor_invalid(trf)	traffic.c

traffic_add_pos

This function adds a new slot to the traffic display for the radio button in the indicated position. The function call is Boolean traffic_add_pos(trf, position).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
position	int	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
False	Boolean	No room
True	Boolean	

Calls	
Function	**Where Described**
traffic_new_height(trf)	traffic.c
radio_name(rp)	radmon.c

traffic_delete_pos

This function deletes the slot for the radio button at the indicated position. The function call is void traffic_delete_pos(trf, position).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
position	int	See Appendix.

Calls	
Function	**Where Described**
traffic_new_height(trf)	traffic.c

traffic_create_radio_button

traffic_create_radio_button adds a new radio selection button to the button list. It is called when a new radio appears. The function call is int traffic_create_radio_button(trf, rp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
rp	RadioInfoP	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
num	int	Button already exists

Calls	
Function	**Where Described**
radio_name(rp)	radmon.c

traffic_create_tune_button

traffic_create_tune_button adds a new tuning button to the button list. It is called when a new radio tuning appears. The function call is int traffic_create_tune_button(trf, tune).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
tune	TUNEP	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
num	int	Button already exists

traffic_reset

traffic_reset resets the traffic display back to its ground state. The function call is void traffic_reset().

Calls	
Function	**Where Described**
traffic_cursor_invalid(trf)	traffic.c
traffic_set_scrollbar(trf)	traffic.c
traffic_erase_drawing(trf)	traffic.c
traffic_measure_grid(trf)	traffic.c
traffic_unfreeze(w, trf, cback)	traffic.c

traffic_new_tune

traffic_new_tune adds a new tuning button. The function call is void traffic_new_tune(tune).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tune	TUNEP	See Appendix.

Calls	
Function	**Where Described**
traffic_create_widget(trf)	traffic.c
traffic_create_tune_button(trf, tune)	traffic.c

traffic_new_radio

traffic_new_radio adds a new radio selection button. The function call is void traffic_new_radio(rp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.

Calls	
Function	**Where Described**
traffic_create_radio_button(trf, rp)	traffic.c

traffic_update

traffic_update updates the traffic display and create the traffic widget, if necessary. It scrolls if necessary and gives permission to keep the present time on the screen. The function call is void traffic_update().

Calls	
Function	**Where Described**
traffic_set_scrollbar(trf)	traffic.c
traffic_left(trf, delta, limit)	traffic.c
traffic_create_widget(trf)	traffic.c

UpdateRadioTraffic

UpdateRadioTraffic adds new radios if necessary and keeps the traffic display up to date. The function call is void UpdateRadioTraffic().

Calls	
Function	**Where Described**
traffic_update()	traffic.c
traffic_new_radio(rp)	traffic.c

traffic_destroy

traffic_destroy destroys resources allocated for the traffic display. The function call is void traffic_destroy().

Calls	
Function	**Where Described**
traffic_delete_pos(trf, position)	traffic.c

tune_selected

tune_selected tests if the specified tuning is currently selected. The buttons given by "bi" are scanned looking for the one that matches the tuning and returning its selected state. The function call is Boolean tune_selected(bi, tune).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
bi	BUTTONINFOP	See Appendix.
tune	TUNEP	See Appendix.

Return Values		
Return Value	**Type**	
pbool	pointer to Boolean	
False	Boolean	

Time CSU

The Time CSU contains code for controlling time functions on the traffic display.

traffic_cursor_valid

traffic_cursor_valid makes the current traffic cursor valid. The status widgets are updated to reflect the new time. The function call is void traffic_cursor_valid(trf).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.

traffic_cursor_invalid

traffic_cursor_invalid makes the current traffic cursor invalid. The status widgets are updated to reflect the present time. The function call is void traffic_cursor_invalid(trf).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.

traffic_set_cursor_time

traffic_set_cursor_time moves the traffic cursor to a specific time. The traffic display is frozen if it was not previously. The displayed is scrolled as necessary to bring the selected time into view. The function call is void traffic_set_cursor_time(time).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
time	MSEC	See Appendix.

Calls	
Function	**Where Described**
traffic_cursor_valid(trf)	traffic.c
traffic_set_scrollbar(trf)	traffic.c
traffic_draw_curosr(trf)	traffic.c
traffic_erase_cursor(trf)	traffic.c
traffic_left(trf, delta, limit)	traffic.c
traffic_right(trf, delta)	traffic.c
traffic_freeze(w, trf, cback	traffic.c
TIME_TO_WINDOW_COORD(trf, t)	traffic.c

traffic_cursor_time

traffic_cursor_time returns the time of the current traffic cursor. If the cursor is not valid, the present time (now) is returned and the function value is FALSE. The function call is BOOLEAN traffic_cursor_time(p).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
p	pointer to MSEC	See Appendix.

Return Values		
Return Value	**Type**	
True	Boolean	
False	Boolean	

traffic_format_time

traffic_format_time constructs a label for a given time. tr_grid_first_format and tr_grid_last_format give the range of fields required for the current scaling. These values essentially discard high-order fields, which are always zero in the current scaling, and low-order fields, which are not necessary to resolve the grid markings. Thus, if the display must represent non-zero hours and the grid markings are multiples of 5 seconds, the tr_grid_first_format will be DATE_HOURS and tr_grid_last_format will be DATE_SECONDS.

The first_formats array gives the format of the first field. Generally, this discards leading zeros. The middle_formats array gives the format of intermediate fields. Generally, this retains all leading zeros in the field. The final_formats array gives the format of the last field. Generally, this includes suffixes which disambiguate the result (e.g., hh:mm is augmented to hh:mm:00 to distinguish if from mm:ss). The only_formats array combines the first_formats and last_formats.

The function call is char * traffic_format_time(trf, time).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trf	TRAFFICINFOP	See Appendix.
time	long	See Appendix.

Return Values		
Return Value	**Type**	
label	char	

traffic_format_traffic_time

This function formats a time value for the main traffic widget. The function call is char * traffic_format_traffic_time(time).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
time	MSEC	See Appendix.

Return Values		
Return Value	**Type**	
result	char	

Calls	
Function	**Where Described**
traffic_format_time(trf, time)	traffic.c

traffic_time_to_date_field

traffic-Time_to_date_field returns the ceiling date type for the specified time. The function call is DATE_FIELDS traffic_time_to_date_field(time).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
time	long	See Appendix.

Return Values		
Return Value	**Type**	
DATE_DAYS	DATE_FIELDS	
DATE_HOURS	DATE_FIELDS	
DATE_MINUTES	DATE_FIELDS	
DATE_SECONDS	DATE_FIELDS	
DATE_M	DATE_FIELDS	
DATE_0M	DATE_FIELDS	
DATE_00M	DATE_FIELDS	

Selection CSU

traffic_sub_select

traffic_sub_select is the callback for traffic selection buttons. It adds and deletes radios from the display as indicated by the selections. The function call is void traffic_sub_select(w, bi, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
bi	BUTTON_INFOP	See Appendix.
cback	pointer to XmListCallbackStruct	See Appendix.

Calls	
Function	**Where Described**
traffic_erase_labels(trf, first_slot, last_slot)	traffic.c
traffic_erase_drawing(trf)	traffic.c
traffic_add_pos(trf, position)	traffic.c
traffic_delete_pos(trf, position)	traffic.c

3.2.3.4 Statistics CSC (state.c)

The Statistics CSC contains code for presenting a summary of radio activity in the State Display. Its routines manipulate states and compile statistics for those states. When the "State" button is selected, a three sub-function buttons appear:

- The *Radio* button presents radio selection buttons. Summary information is presented for the radios selected from this panel.
- The *Attributes* button presents attribute selection buttons. Selected attributes determine which states are included in the summary.
- The *Status* button presents status type selection buttons.

The state display consists of one or more rows of information, which rsummarize one or more staradios. Each column contains the value of an attribute or statistic for those states. Statistics include:

- The percentage of all time spent in the state
- The number of times a radio was in this state.
- The average duration of time spent in this state.
- The maximum duration of time spent in this state.

The State Display CSC consists of seven CSUs, shown in Figure 3.2.3.4-1.

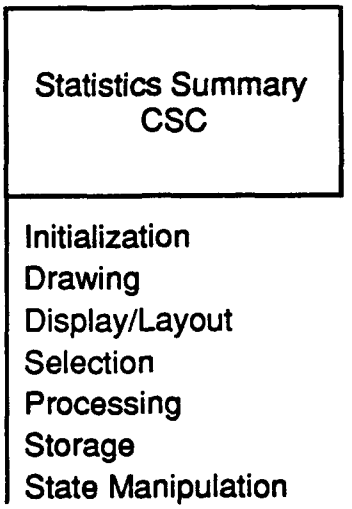


Figure 3.2.3.4-1. GUI—State Display CSC Structure

Initialization CSU

The Initialization CSU contains code for initializing the State Display.

stateReplenishFreeList

This function preallocates storage for interrupt routine use because interrupt routines cannot allocate memory. The function call is void stateReplenishFreeList().

Calls	
Function	**Where Described**
DisablePeriodic()	radmon.c

stateAllocate

This function allocates a state_vector entry. The function call is STATEP stateAllocate().

Return Values		
Return Value	**Type**	
state_vector()	STATEP	

state_init

state_init initializes state things. The function call is void state_init().

Calls	
Function	**Where Described**
htInit(size, key)	hTable.c
stateReplenishFreeList()	state.c

state_close_callback

state_close_callback activates the callback for the close button on the state display and removes the state display from the screen. The function call is void state_close_callback().

stateRegisterNames

stateRegisterNames registers names for Mrm. The function call is void stateRegisterNames().

stateInitWidgets

stateInitWidgets gets widget ids for various state widgets. The function call is void stateInitWidgets().

Calls	
Function	**Where Described**
NameToWidget(parent, name)	display.c

Drawing CSU

state_draw_s_com

state_draw_s_com draws an S_COM (state component). This is basically a string. The com_max argument gives width information to achieve a columnar format. The function call is void state_draw_s_com(d, w, xp, y, com, com_max, left_justify).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
d	pointer to Display	See Appendix.

w	Window	See Appendix.
xp	pointer to Position	See Appendix.
y	Position	See Appendix.
com	S_COM	See Appendix.
com_max	S_COM	See Appendix.
left_justify	Boolean	See Appendix.

state_draw_one_com

state_draw_one_com draws one line of the state display. rc and dc give the S_COMs to be drawn. The function call is void state_draw_one_com(d, w, y, rc, dc).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
d	pointer to Display	See Appendix.
w	Window	See Appendix.
y	Position	See Appendix.
rc	RADIO_COMP	See Appendix.
dc	DERIVED_COMP	See Appendix.

Calls	
Function	**Where Described**
state_draw_s_com(d, w, sp, y, com, com_max, left_justify)	state.c

state_expose_callback

state_expose_callback exposes the callback for the state display and loops through all states, drawing the distinct ones. The function call is void state_expose_callback().

Calls	
Function	**Where Described**
state_draw_one_com(d, w, y, rc, dc)	state.c

state_clear_dci

state_clear_dci clears the st_dci field of all states. The function call is void state_clear_dci().

Display/Layout CSU

state_new_radio_buttons

state_new_radio_buttons creates new radio buttons. The function call is void state_new_radio_buttons().

Calls	
Function	**Where Described**
radio_name(rp)	radmon.c

state_create_widget

state_create_widget creates a new state widget. The function call is void state_create_widget().

Calls	
Function	**Where Described**
NameToWidget(parent, name)	display.c

state_format_status

state_format_status formats the "status," "from," and "at" fields for a state. The function call is void state_format_status(st, sc).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
st		See Appendix.
sc	STATE_COMP	See Appendix.

Calls	
Function	**Where Described**
state_finish_com(cm, max_cm, s)	state.c
state_radio_name(st)	state.c

state_format_tuning

state_format_tuning formats the tuning and TOD fields of the state display. The function call is state_format_tuning(tune, sc).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tune	TUNEP	See Appendix.
sc	STATE_COMP	See Appendix.

Calls	
Function	**Where Described**
state_finish_com(cm, max_cm, s)	state.c

state_format

state_format format the fields of a state. The function call is state_format(sc).

Calls	
Function	**Where Described**
state_finish_com(cm, max_cm, s)	state.c
state_format_tuning(tune, sc)	state.c
state_format_status(st, sc)	state.c

state_format_statistics

state_format_statistics formats the statistics components of the state line. The function call is int state_format_statistics(dc, n, overall_time).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
dc	DERIVED_COMP	See Appendix.
n	int	See Appendix.
overall_time	MSEC	See Appendix.

Return Values		
Return Value	**Type**	
n_distinct	int	

Calls	
Function	**Where Described**
state_is_distinct(st1, st2)	state.c
state_finish_com(cm, max_cm, s)	state.c

state_display

state_display computes a new state display. The function call is void state_display(force).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
force	Boolean	See Appendix.

Calls	
Function	**Where Described**
state_compile_statistics(rc, total_time)	state.c
state_expand_radio_com(n_radios)	state.c
state_new_radio_buttons()	state.c
state_create_widget()	state.c
state_finish_com(cm, max_cm, s)	state.c
state_format(sc)	state.c
state_expand_state_comp(n)	state.c
state_clear_dci()	state.c
state_fill_one_radio(rc, rp, state_now)	state.c

state_function

state_function activates the callback for the state button and generates the state display. The function call is void state_function().

Calls	
Function	**Where Described**
state_display(force)	state.c

state_update

state_update updates the state display because new states or new radios exist. The function call is void state_update().

Calls	
Function	**Where Described**
state_display(force)	state.c

Selection CSU

state_attr_select

state_attr_select activates the callback for the attribute button list. Certain buttons are coupled together; therefore, selecting certain buttons forces other buttons to be selected as well, and deselecting certain buttons forces other buttons to be deselected. The procedure is:

1. Build a mask of the filter bits for all selected buttons.
2. Compute the mask of buttons which became unselected (clr_bits).
3. Compute the mask of buttons which became selected (set_bits).
4. Compute the mask of buttons which should also be selected by or'ing together the set_coupled_bits for the newly selected buttons.
5. Similarly, compute the mask of buttons which should also be unselected by or'ing together the clr_coupled_bits for the newly deselected buttons.
6. Augment clr_bits by the also_clr bits and augment set_bits by the also_set bits.
7. Remove any bits being cleared from the bits being set.
8. Change the selection state of the buttons which are also_clr and also_set.
9. Remove the clr_bits from state_filter and add set_bits.

The function call is void state_attr_select(w, a, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
a	caddr_t	See Appendix.
cback	pointer to XmButtonLCallbackStruct	See Appendix.

Calls	
Function	**Where Described**
state_display(force)	state.c

state_status_select

state_status_select activates the callback for the status selection buttons and computes the new state_status_filter from the selected buttons. The function call is void state_status_select(w, a, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
a	caddr_t	See Appendix.
cback	pointer to XmButtonLCallbackStruct	See Appendix.

Calls	
Function	**Where Described**
state_display(force)	state.c

state_radio_select

state_radio_select changes for which radios the state display is generated. state_display tests the selected status of the radio selection buttons directly, so this function only calls state_display to regenerate the display according to the new set of selected radios. The function call is void state_radio_select(w, a, cback).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	Widget	See Appendix.
a	caddr_t	See Appendix.
cback	pointer to XmButtonLCallbackStruct	See Appendix.

Calls	
Function	**Where Described**
state_display(force)	state.c

Processing CSU

state_fill_one_radio

state_fill_one_radio processes the states of one radio and computes the derived state information. All the states the radio has ever been in are examined. The first time a state is examined, the corresponding derived information is initialized and attached to the state. In any case, the derived information is accumulated according to how many times the state is entered and how long the radio is in that state. The function call is void state_fill_one_radio(rc, rp, state_now).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rc	RADIO_COMP	See Appendix.
rp	RadioInfoP	See Appendix.
state_now	MSEC	See Appendix.

Calls	
Function	**Where Described**
state_expand_derived_com(rc, n)	state.c

state_compile_statistics

state_compile_statistics formats the statistics for a radio (or radios), computes the radio field, and returns the number of distinct states. The function call is int state_compile_statistics(rc, total_time).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rc	RADIO_COMP	See Appendix.
total_time	MSEC	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
n_distinct	int	Number of distinct states

Calls	
Function	**Where Described**
state_finish_com(cm, max_cm, s)	state.c
state_format_statistics(dc, n, overall_time)	state.c
radio_name(rp)	radmon.c

state_com_compare

state_com_compare compares two RADIO_COMPs for sorting. The function call is int state_radio_com_compare(rc1, rc2).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rc1, rc2	RADIO_COMP	See Appendix.

Return Values		
Return Value	**Type**	
-1	int	
1	int	
0	int	

state_derived_com_compare

state_derived_com_compare compares two DERIVED_COMs for sorting. The function call is int state_derived_com_compare(dc1, dc2).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
dc1,dc2	DERIVED_COMP	See Appendix.

Return Values		
Return Value	**Type**	
-1	int	
1	int	
0	int	

Calls	
Function	**Where Described**
tune_compare(t1, t2)	tune.c

state_is_distinct

state_is_distinct tests whether two states are distinct given the current state_filter. The function call is Boolean state_is_distinct(st1, st2).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
st1,st2	STATEP	See Appendix.

Return Values		
Return Value	**Type**	
False	Boolean	
True	Boolean	

state_expand_state_comp

state_expand_state_comp increases the size of the state_state_comp vector to accommodate more states. The function call is void state_expand_state_comp(n).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
n	int	See Appendix.

state_expand_derived_com

state_expand_derived_com increases the size of a DERIVED_COM vector. The function call is void state_expand_derived_com(rc, n).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rc	RADIO_COMP	See Appendix.
n	int	See Appendix.

state_expand_radio_com

state_expand_radio_com increases the size of the state_radio_com vector. The function call is void state_expand_radio_com(n_radios).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
n_radios	int	See Appendix.

Storage CSU

state_finish_com

state_finish_com stores a string as the value of an S_COM. It reallocates the space for the value if necessary and updates the maximum width in max_cm as needed. The function call is void state_finish_com(cm, max_cm, s).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
cm	register S_COMP	See Appendix.
max_cm	S_COMP	See Appendix.
s	pointer to char	See Appendix.

state_destroy

state_destroy releases state related storage. The function call is void state_destroy().

Calls	
Function	**Where Described**
htInit(size, key)	hTable.c
htFree(ht)	hTable.c
stateReplenishFreeList()	hTable.c

State Manipulation CSU

state_count_file

state_count_file counts how many lines of output will be written for the state section of a radmon file. The function call is int state_count_file().

Return Values		
Return Value	**Type**	
state_n_vector+1	int	

state_write_file

state_write_file writes the state section of a radmon file. The function call is void state_write_file(f).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
f	pointer to FILE	See Appendix.

Calls	
Function	**Where Described**
file_count(n)	file.c

state_read_file

state_read_file reads the state section of a radmon file. The function call is char *state_read_file(f, version).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
version	int	See Appendix.

Return Values		
Return Value	**Type**	
"Premature EOF while reading %s"	char	
"File %s has bad format"	char	
NULL		

Calls	
Function	**Where Described**
htReplenish()	hTab le.c
stateReplenishFreeList()	state.c
state_find(st)	state.c
tune_nth(n)	tune.c
file_gets(buf, size, f)	file.c

state_find

state_find finds the state vector entry for the given state. If the state is not found, an entry is made. The function call is STATEP state_find(st).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
st	STATEP	See Appendix.

Return Values		
Return Value	**Type**	
value	STATEP	

Calls	
Function	**Where Described**
htInsert(ht, name, value)	hTable.c
htFind(ht, name, value)	hTable.c
stateAllocate()	state.c

state_nth

state_nth returns the nth state from the state_vector. The function call is STATEP state_nth(n).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
n	int	state.c

Return Values		
Return Value	**Type**	
state_vector[n]	array of STATEP	

3.2.4 Utilities CSC

The Utilities CSC provides utilities used by the other CSCs. It is comprised of three CSUs, as shown in Figure 3-2.4-1.

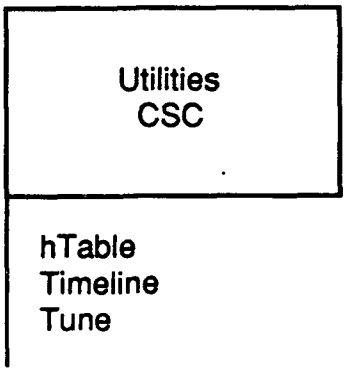


Figure 3-2.4-1. Utilities CSC Structure

hTable CSU

The hTable CSU consists of hash table routines.

htReplenish

htReplenish replenishes storage for hashentries. htInsert may be called from interrupt level and is not allowed to allocate memory. We preallocate here and keep the entries on a free list. The function call is void htReplenish().

htFreeInit

htFreeInit initializes the free list and counts and does the initial replenishment. The function call is void htFreeInit().

Calls	
Function	**Where Described**
htReplenish	hTable.c

hash()

hash() computes a hash index from the key. It combines the bytes making up the key into an integer index into the hash table. The function call is int hash(name, ht).

Return Values		
Return Value	**Type**	
-n	int	
n	int	

htInit()

htInit(initializes a hash table and creates a hash table of the specified size and key size. The function call is HTABLEP htInit(size, key).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
size	int	See Appendix.
key	int	See Appendix.

Return Values		
Return Value	**Type**	
ht	HTABLEP	

htFree

htFree releases hash table storage. The function call is void htFree(ht).

htInsert

htInsert inserts an entry in the hash table. It searches for an existing entry matching the key. If found, its value is changed; otherwise, a new entry is created and chained onto the list for the proper bucket. In either case, the HASHENTRY is returned. The function call is HASHENTRYP htInsert(ht, name, value).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ht	HTABLEP	See Appendix.
name	pointer to char	See Appendix.
value	int	See Appendix.

Return Values		
Return Value	**Type**	
p	HASHENTRYP	

Calls	
Function	**Where Described**
hash(name, ht)	hTable.c

htFind

htFind searches the hash table for the specified key and returns the HASHENTRY found. It returns NULL if the entry cannot be found. The function call is HASHENTRYP htFind(ht, name, value).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ht	HTABLEP	See Appendix.
name	pointer to char	See Appendix.
value	int	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
p	HASHENTRYP	
null	HASHENTRYP	Name not found, return failure.

Calls	
Function	**Where Described**
hash(name, ht)	hTable.c

Timeline CSU

The Timeline CSU consists of routines for manipulating timelines.

tlEnqueue

tlEnqueue adds a segment to the indicated queue. The function call is tlEnqueue(tlq, tls).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tlq	TLQP	See Appendix.
tls	TLSP	See Appendix.

tlDequeue

tlDequeue removes a segment from the indicated queue. The function call is TLSP tlDequeue(tlq).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tlq	TLQP	See Appendix.

Return Values		
Return Value	**Type**	
tls	TLSP	

tlReplenishFreeList

tlReplenishFreeList replenishes the free list. Interrupt level routines can't allocate memory, so we keep a list of available segments to be used at interrupt level. The function call is void tlReplenishFreeList().

Calls	
Function	**Where Described**
tlEnqueue(tlq, tls)	timeline.c
DisablePeriodic()	radmon.c

tlInit

tlInit initializes timeline things. The function call is void tlInit().

Calls	
Function	**Where Described**
tlReplenishFreeList()	timeline.c

tlChangeState

tlChangeState adds a new state transition to a radio. If the current segment is full, another is allocated. The function call is void tlChangeState(rp, timestamp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.
timestamp	MSEC	See Appendix.

Calls	
Function	**Where Described**
tlEnqueue	timeline.c
tlDequeue	timeline.c

\

tlNext

tlNext advances the given segment and element pointers to the next element. It returns false if there are no more. The function call is Boolean tlNext(tlsp, tlep).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tlsp	pointer to TLSP	See Appendix.
tlep	pointer to TLEP	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
False	Boolean	No more elements
True	Boolean	Advances segment and element pointer to next element.

tlPrev

tlPrev steps the given segment and element pointers to the previous element. The function call is Boolean tlPrev(tlsp, tlep).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tlsp	pointer to TLSP	See Appendix.
tlep	pointer to TLEP	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
True	Boolean	Steps segment and element pointers to previous element.
False	Boolean	No previous element

tlFind

tlFind finds the element which embraces the given time and returns its segment and element pointers. Returns False if there is no such element. The function call is Boolean tlFind(rp, when, tlsp, tlep).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.
when	long	See Appendix.
tlsp	pointer to TLSP	See Appendix.
tlep	pointer to TLEP	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
True	Boolean	Time element segment and element pointers
False	Boolean	No such element

tlReadFile

tlReadFile reads a timeline portion of a radmon file. The function call is char *tlReadFile(f, rp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.
f	pointer to FILE	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
"Premature EOF while reading %s"	char	Error message
"File %s has bad format"	char	Error message
NULL		No error found (?)

Calls	
Function	**Where Described**
tlEnqueue(tlq, tls)	timeline.c
tlDequeue(tlq)	timeline.c
tlReplenishFreeList()	timeline.c
file_gets(buf, size, f)	file.c
state_nth(n)	state.c

tlCountFile

tlCountFile counts haow many lines are required to write a timeline. The function call is int tlCountFile(rp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.

Return Values		
Return Value	**Type**	
rp-> r_tl_queue.tlq_n_entries + 1	int	

tlWriteFile

tlWriteFile writes a timeline portion of a radmon file. The function call is void tlWriteFile(f, rp).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	RadioInfoP	See Appendix.
f	pointer to FILE	See Appendix.

Calls	
Function	**Where Described**
file_count(n)	file.c

tlFree

tlFree releases timeline information for a radio. The function call is void tlFree(rp).

Calls	
Function	**Where Described**
tlEnqueue(tlq, tls)	timeline.c
tlDequeue(tlq)	timeline.c

Tune CSU

The Tune CSU consists of routines to keep track of tunings.

tune_count_file

tune_count_file counts the number of output lines for the tuning section of a radmon file. The function call is int tune_count_file().

Return Values		
Return Value	**Type**	**Meaning**
tune_n_list+ 1	int	Number of output lines for tuning section of a radmon file.

tune_write_file

tune_write_file writes the tuning section of a radmon file. The function call is void tune_write_file(f).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
f	pointer to FILE	See Appendix.

Calls	
Function	**Where Described**
file_count(n)	file.c

tune_read_file

tune_read_file reads the tuning section of a radmon file. The function call is char *tune_read_file(f, version).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
f	pointer to FILE	See Appendix.
version	int	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
"Premature EOF while reading %s	char	Error message
"File %s has bad format"	char	Error message
NULL		No error

Calls	
Function	**Where Described**
htReplenish()	hTable.c
tuneReplenishFreeList()	tune.c
tune_find_mode, frequency, hopinfo)	tune.c
file_gets(bug, size, f)	file.c

tuneReplenishFreeList

tuneReplenishFreeList replenishes the list of free tunings. We preallocate tunings and keep them in a list so interrupt level code doesn't have to allocate storage. The function call is void tuneReplenishFreeList().

Calls	
Function	**Where Described**
DisablePeriodic()	radmon.c

tuneAllocate

tuneAllocate gets a tuning from the free list. The function call is TUNEP tuneAllocate().

Return Values		
Return Value	**Type**	
tune	TUNEP	

tune_init

tune_init initializes tuning stuff and performs initial replenishment. The function call is void tune_init().

Calls	
Function	**Where Described**
htInit(size, key)	hTable.c
tuneReplenishFreeList()	tune.c

tune_set_name

tune_set_name sets the q_name of a tuning from the frequency or hopset id. We do this once when the tuning is created so the name is readily available. The function call is void tune_set_name(tune1). .

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tune1	TUNEP	See Appendix.

tune_find

tune_find finds a tuning for the given mode and frequency or hopinfo. If the tuning does not already exist, it is created. The function call is TUNEP tune_find(mode, frequency, hopinfo).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mode	unsigned char	See Appendix.
frequency	long	See Appendix.
hopinfo	pointer to FHParameters	See Appendix.

Return Values		
Return Value	**Type**	
value	TUNEP	

Calls	
Function	**Where Described**
tune_set_name(tune1)	tune.c
htInsert(ht, name, value)	hTable.c
htFind(ht, name, value)	hTable.c
tuneAllocate()	tune.c

tune_new

tune_new is called when new tunings have been created to update the tune_list and create tuning buttons for the traffic display. The function call is void tune_new().

Calls	
Function	**Where Described**
traffic_new_tune(tune)	traffic.c

tune_compare

tune_compare compares two tunings. It is used by the state display to sort states. The function call is int tune_compare(t1, t2).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
t1	TUNEP	See Appendix.
t2	TUNEP	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
0	int	Tunings are equal.
-1	int	
1	int	

tune_destroy

tune_destroy releases tuning storage. The function call is void tune_destroy().

Calls	
Function	**Where Described**
htInit(size, key)	hTable.c
htFree(ht)	hTable.c

tune_nth

tune_nth returns the nth tuning. The function call is TUNEP tune_nth(n).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
n	int	See Appendix.

Return Values		
Return Value	**Type**	**Meaning**
tune_list[n]	TUNEP	nth tuning

Appendix:

Radio Performance Monitor (RADMON) CSCI

Typedef Reference

Table of Contents

FILE: display.c

FUNCTION: quit_callback(w)

parameter: Widget w, defined in <X11/Intrinsic.h>

FUNCTION: dump_callback(w)

parameter: Widget w, defined in <X11/Intrinsic.h>

FUNCTION: reset_callback(w)

parameter: Widget w, defined in <X11/Intrinsic.h>

FUNCTION: function_descend(fbc, fn)

parameter: FUNCTION_BUTTON_CLOSUREP fbc, defined in radmon.h

parameter: void (*fn)(), standard type

FUNCTION: display_deselect()

FUNCTION: function_callback(w, fbc, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: FUNCTION_BUTTON_CLOSUREP fbc, defined in radmon.h

parameter: XmButtonLCallbackStruct *cback

FUNCTION: create_function_items(w, fbc)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: FUNCTION_BUTTON_CLOSUREP fbc, defined in radmon.h

FUNCTION: clipResize(w, data, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: caddr_t data, defined in <sys/types.h>

parameter: XmAnyCallbackStruct, standard type

FUNCTION: NameToWidget(parent, name)

parameter: Widget parent, defined in <X11/Intrinsic.h>

parameter: char *name, standard type

FUNCTION: CvtStringToWidget(args, nargs, fromVal, toVal)

parameter: XrmValuePtr args, fromVal, toVal, defined in <X11/Xresource.h>

parameter: int *nargs, standard type

FUNCTION: CvtStringToXmStringLtoR(args, nargs, fromVal, toVal)

parameter: XrmValuePtr args, fromVal, toVal, defined in <X11/Xresource.h>

parameter: int *nargs, standard type

FUNCTION: CvtStringToColor(args, nargs, fromVal, toVal)

parameter: XrmValuePtr args, fromVal, toVal, defined in <X11/Xresource.h>

parameter: int *nargs, standard type

FUNCTION: display_init(argcp, argv)

parameter: int *argcp, standard type

parameter: char *argv[], standard type

FUNCTION: Dispatch()

FILE: file.c

FUNCTION: file_ok(w, data, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: caddr_t data, defined in <sys/types.h>

parameter: XmSelectionBoxCallbackStruct *cback, defined in <Xm/Xm.h>

FUNCTION: file_popup(w)

parameter: Widget w, defined in <X11/Intrinsic.h>

FUNCTION: file_error(fmt, arg)

parameter: char *fmt, standard type

parameter: char *arg, standard type

FUNCTION: file_flush()

FUNCTION: file_inform(fmt, arg, maximum)

parameter: char *fmt, standard type

parameter: char *arg, standard type

parameter: int maximum, standard type

FUNCTION: file_inform_end()

FUNCTION: file_gets(buf, size, f)

parameter: char *buf, standard type

parameter: int size, standard type

parameter: FILE *f, defined in <stdio.h>

FUNCTION: file_count(n)

parameter: int n, standard type

FUNCTION: file_write_file(filename)

parameter: char *filename, standard type

FUNCTION: file_write()

FUNCTION: file_read_file(filename)

parameter: char *filename, standard type

FUNCTION: file_read()

FUNCTION: fileRegisterNames()

FUNCTION: fileInitWidgets()

FILE: hTable.c

FUNCTION: htReplenish()

FUNCTION: htFreeInit()

FUNCTION: hash(name, ht)

parameter: char *name, standard type

parameter: HTABLEP ht, defined in hTable.h

FUNCTION: htInit(size, key)

parameter: int size, standard type

parameter: int key, standard type

FUNCTION: htFree(ht)

parameter: HTABLEP ht, defined in hTable.h

FUNCTION: htInsert(ht, name, value)

parameter: HTABLEP ht, defined in hTable.h

parameter: char *name, standard type

parameter: int value, standard type

FUNCTION: htFind(ht, name, value)

parameter: HTABLEP ht, defined in hTable.h

parameter: char *name, standard type

parameter: hash_value_type *value, defined in hTable.h

FILE: map.c

FUNCTION: ForegroundColorDefault(widget, offset, valPtr)

parameter: Widget widget, defined in <X11/Intrinsic.h>

parameter: int offset, standard type

parameter: XrmValuePtr valPtr, defined in <X11/Xresource.h>

FUNCTION: BackgroundColorDefault(widget, offset, valPtr)

parameter: Widget widget, defined in <X11/Intrinsic.h>

parameter: int offset, standard type

parameter: XrmValuePtr valPtr, defined in <X11/Xresource.h>

FUNCTION: CvtStringToMapStyle(args, nargs, from, to)

parameter: XrmValue *args, defined in <X11/Xresource.h>

parameter: int *nargs, standard type

parameter: XrmValue *from, defined in <X11/Xresource.h>

parameter: XrmValue *to, defined in <X11/Xresource.h>

FUNCTION: map_select(w, a, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: caddr_t a, defined in <sys/types.h>

parameter: XmListCallbackStruct *cback, defined in <Xm/Xm.h>

FUNCTION: map_time_callback(w, a, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: caddr_t a, defined in <sys/types.h>

parameter: XmAnyCallbackStruct *cback, defined in <Xm/Xm.h>

FUNCTION: map_color_animate()

FUNCTION: map_style_of_type(type)

parameter: unsigned char type, standard type

FUNCTION: map_shape_of_types(type1, type2)

parameter: unsigned char type1, type2, standard type

FUNCTION: map_gc_of_type(type, lesser_type, gcp1, gcp2)

parameter: unsigned char type, standard type

parameter: unsigned char lesser_type, standard type

parameter: GC *gcp1, *gcp2, defined in <X11/Xlib.h>

FUNCTION: map_bounding_box(pts, npts)

parameter: register XPoint *pts, defined in <X11/Xlib.h>

parameter: int npts, standard type

FUNCTION: map_draw_bolt(dpy, wind, rp1, rp2, type1, type2, shape, mode)

parameter: Display *dpy, defined in <X11/Xlib.h>

parameter: Window wind, defined in <X11/X.h>

parameter: RadioInfoP rp1, rp2, defined in radmon.h

parameter: unsigned char type1, type2, standard type

parameter: unsigned char shape, standard type

parameter: DRAW_MODE mode, defined in this file

FUNCTION: map_reset_pending_draw()

FUNCTION: map_draw()

FUNCTION: map_start_draw()

FUNCTION: map_erase()

FUNCTION: map_start_erase()

FUNCTION: map_color_radio(rp)

parameter: RadioInfoP rp, defined in radmon.h

FUNCTION: map_color_vehicle(vp)

parameter: VehicleInfoP vp, defined in radmon.h

FUNCTION: map_display_move(visible, x, y)

parameter: Boolean visible, defined in radmon.h

parameter: Position x, y, defined in <X11/Intrinsic.h>

FUNCTION: map_input(w, vp, event)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: VehicleInfoP vp, defined in radmon.h

parameter: XEvent *event, defined in <X11/Xlib.h>

FUNCTION: map_expose(w, a, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: caddr_t a, defined in <sys/types.h>

parameter: XmDrawingAreaCallbackStruct *cback, defined in <Xm/Xm.h>

FUNCTION: map_resize(w)

parameter: Widget w, defined in <X11/Intrinsic.h>

FUNCTION: map_close()

FUNCTION: map_step()

FUNCTION: map_step_to_anomaly()

FUNCTION: map_freeze()

FUNCTION: map_unfreeze()

FUNCTION: mapRegisterNames()

FUNCTION: mapInitWidgets()

FUNCTION: map_allocate_colors()

FUNCTION: map_animation_pixmap(dir, rootwindow)

parameter: int dir, standard type

parameter: Window rootwindow, defined in <X11/X.h>

FUNCTION: map_create_widget()

FUNCTION: map_create_vehicle_widget(vp)

parameter: VehicleInfoP vp, defined in radmon.h

FUNCTION: map_remanage_vehicle(vp)

parameter: VehicleInfoP vp, defined in radmon.h

FUNCTION: map_layout_compare_x(vpp1, vpp2)

parameter: VehicleInfoP *vpp1, *vpp2, defined in radmon.h

FUNCTION: map_layout_compare_y(vpp1, vpp2)

parameter: VehicleInfoP *vpp1, *vpp2, defined in radmon.h

FUNCTION: map_measure()

FUNCTION: map_layout()

FUNCTION: map_create_radio_buttons()

FUNCTION: map_display()

FUNCTION: map_set_time(t)

parameter: MSEC t, defined in timeline.h

FUNCTION: map_recolor()

FUNCTION: map_update()

FUNCTION: map_set_connectivity(rp1, idx, new_connectivity)

parameter: RadioInfoP rp1, defined in radmon.h

parameter: int idx, standard type

parameter: unsigned char new_connectivity, standard type

FUNCTION: map_vehicle_moved(vp)

parameter: VehicleInfoP vp, defined in radmon.h

FUNCTION: map_more_radios()

FUNCTION: map_vehicle_vanished(vp1)

parameter: VehicleInfoP vp1, defined in radmon.h

FUNCTION: map_vehicle_appeared(vp1)

parameter: VehicleInfoP vp1, defined in radmon.h

FUNCTION: map_init_radio(rp)

parameter: RadioInfoP rp, defined in radmon.h

FUNCTION: map_reset()

FUNCTION: map_radio_transmit_time(rp, when)

parameter: RadioInfoP rp, defined in radmon.h

parameter: MSEC when, defined in timeline.h

FUNCTION: map_set_radio_time(rp, when)

parameter: RadioInfoP rp, defined in radmon.h

parameter: MSEC when, defined in timeline.h

FUNCTION: map_advance(up_to, advance_mode, max_advance)

parameter: MSEC up_to, defined in timeline.h

parameter: MAP_ADVANCE_MODE advance_mode, defined in this file

parameter: MSEC max_advance, defined in timeline.h

FUNCTION: map_advance_display(up_to, max_advance, if_frozen)

parameter: MSEC up_to, defined in timeline.h

parameter: MSEC max_advance, defined in timeline.h

parameter: Boolean if_frozen, defined in radmon.h

FUNCTION: map_destroy()

FILE: net.c

FUNCTION: InitSimNetwork()

FUNCTION: net_destroy()

FUNCTION: VehicleIDtoIndex(vehicleID)

parameter: VehicleID vehicleID

FUNCTION: net_read_file(f, version)

parameter: FILE *f, defined in <stdio.h>

parameter: int version, standard type

FUNCTION: net_count_file(f)

parameter: FILE *f, defined in <stdio.h>

FUNCTION: net_write_file(f)

parameter: FILE *f, defined in <stdio.h>

FUNCTION: netReplenish()

FUNCTION: ReadPDUs()

FUNCTION: DrainPDUs()

FUNCTION: netChangeState(rp, st, timestamp)

parameter: RadioInfoP rp, defined in radmon.h

parameter: STATEP st, defined in state.h

FUNCTION: ProcessVehicleAppearancePDU(pdu, timestamp)

parameter: VehicleAppearanceVariant *pdu

parameter: MSEC timestamp, defined in timeline.h

FUNCTION: ProcessDeactivatePDU(pdu, timestamp)

parameter: DeactivateResponseVariant *pdu

parameter: MSEC timestamp, defined in timeline.h

FUNCTION: ProcessStatusChangePDU(pdu, timestamp)

parameter: StatusChangeVariant *pdu

parameter: MSEC timestamp, defined in timeline.h

FUNCTION: ProcessVehicleStatusPDU(pdu, timestamp)

parameter: VehicleStatusVariant *pdu

parameter: MSEC timestamp, defined in timeline.h

FUNCTION: ProcessTransmitterPDU(pdu, timestamp)

parameter: register TransmitterVariant *pdu

parameter: MSEC timestamp, defined in timeline.h

FUNCTION: ProcessReceiverPDU(pdu, timestamp)

parameter: register ReceiverVariant *pdu

parameter: MSEC timestamp, defined in timeline.h

FUNCTION: net_geteaddr()

FILE: radmon.c

FUNCTION: main(argc, argv)

parameter: int argc, standard type

parameter: char **argv, standard type

FUNCTION: slow_periodic()

FUNCTION: medium_periodic()

FUNCTION: EnablePeriodic()

FUNCTION: DisablePeriodic()

FUNCTION: fast_periodic()

FUNCTION: resetTime()

FUNCTION: InitVehicle(vp, vidx)

parameter: VehicleInfoP vp, defined in radmon.h

parameter: int vidx, standard type

FUNCTION: DestroyEverything()

FUNCTION: set_identification_label()

FUNCTION: Offline()

FUNCTION: Online()

FUNCTION: DestroyWidget(wp)

parameter: Widget *wp, defined in <X11/Intrinsic.h>

FUNCTION: DestroyVehicles()

FUNCTION: resetRadios()

FUNCTION: TimeoutRadios()

FUNCTION: TimeoutVehicles()

FUNCTION: vehicle_name(vp)

parameter: VehicleInfoP vp, defined in radmon.h

FUNCTION: radio_name(rp)

parameter: RadioInfoP rp, defined in radmon.h

FUNCTION: state_radio_name(st)

parameter: STATEP st, defined in state.h

FUNCTION: exit_gracefully()

FUNCTION: print_banner()

FILE: state.c

FUNCTION: stateReplenishFreeList()

FUNCTION: stateAllocate()

FUNCTION: state_init()

FUNCTION: state_count_file()

FUNCTION: state_write_file(f)

parameter: FILE *f, defined in <stdio.h>

FUNCTION: state_read_file(f, version)

parameter: FILE *f, defined in <stdio.h>

parameter: int version, standard type

FUNCTION: state_find(st)

parameter: STATEP st, defined in state.h

FUNCTION: state_close_callback()

FUNCTION: state_is_distinct(st1, st2)

parameter: STATEP st1, defined in state.h

parameter: STATEP st2, defined in state.h

FUNCTION: state_draw_s_com(d, w, xp, y, com, com_max, left_justify)

parameter: Display *d, defined in <X11/Xlib.h>

parameter: Window w, defined in <X11/X.h>

parameter: Position *xp, defined in <X11/Intrinsic.h>

parameter: Position y, defined in <X11/Intrinsic.h>

parameter: S_COM com, defined in state.h

parameter: S_COM com_max, defined in state.h

parameter: Boolean left_justify, defined in radmon.h

FUNCTION: state_draw_one_com(d, w, y, rc, dc)

parameter: Display *d, defined in <X11/Xlib.h>

parameter. Window w, defined in <X11/X.h>

parameter: Position y, defined in <X11/Intrinsic.h>

parameter: RADIO_COMP rc, defined in state.h

parameter: DERIVED_COMP dc, defined in state.h

FUNCTION: state_expose_callback()

FUNCTION: state_attr_select(w, a, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: caddr_t a, defined in <sys/types.h>

parameter: XmButtonLCallbackStruct *cback

FUNCTION: state_status_select(w, a, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: caddr_t a, defined in <sys/types.h>

parameter: XmButtonLCallbackStruct *cback

FUNCTION: state_radio_select(w, a, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: caddr_t a, defined in <sys/types.h>

parameter: XmButtonLCallbackStruct *cback

FUNCTION: stateRegisterNames()

FUNCTION: stateInitWidgets()

FUNCTION: state_new_radio_buttons()

FUNCTION: state_create_widget()

FUNCTION: state_finish_com(cm, max_cm, s)

parameter: register S_COMP cm, defined in state.h

parameter: S_COMP max_cm, defined in state.h

parameter: char *s, standard type

FUNCTION: state_format_status(st, sc)

parameter: STATEP st, defined in state.h

parameter: STATE_COMP sc, defined in state.h

FUNCTION: state_format_tuning(tune, sc)

parameter: TUNEP tune, defined in tune.h

parameter: STATE_COMP sc, defined in state.h

FUNCTION: state_format(sc)

parameter: STATE_COMP sc, defined in state.h

FUNCTION: state_expand_state_comp(n)

parameter: int n, standard type

FUNCTION: state_expand_derived_com(rc, n)

parameter: RADIO_COMP rc, defined in state.h

parameter: int n, standard type

FUNCTION: state_clear_dci()

FUNCTION: state_fill_one_radio(rc, rp, state_now)

parameter: RADIO_COMP rc, defined in state.h

parameter: RadioInfoP rp, defined in radmon.h

parameter: MSEC state_now, defined in timeline.h

FUNCTION: state_format_statistics(dc, n, overall_time)

parameter: DERIVED_COMP dc, defined in state.h

parameter: int n, standard type

parameter: MSEC overall_time, defined in timeline.h

FUNCTION: state_compile_statistics(rc, total_time)

parameter: RADIO_COMP rc, defined in state.h

parameter: MSEC total_time, defined in timeline.h

FUNCTION: state_radio_com_compare(rc1, rc2)

parameter: RADIO_COMP rc1, defined in state.h

parameter: RADIO_COMP rc2, defined in state.h

FUNCTION: state_derived_com_compare(dc1, dc2)

parameter: DERIVED_COMP dc1, defined in state.h

parameter: DERIVED_COMP dc2, defined in state.h

FUNCTION: state_expand_radio_com(n_radios)

parameter: int n_radios, standard type

FUNCTION: state_display(force)

parameter: Boolean force, defined in radmon.h

FUNCTION: state_function()

FUNCTION: state_update()

FUNCTION: state_nth(n)

parameter: int n, standard type

FUNCTION: state_destroy()

FILE: status.c

FUNCTION: status_close_callback(w, rp, b)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: RadioInfoP rp, defined in radmon.h

parameter: caddr_t b, defined in <sys/types.h>

FUNCTION: status_select(w, a, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: caddr_t a, defined in <sys/types.h>

parameter: XmButtonLCallbackStruct *cback

FUNCTION: set_widget_label()

FUNCTION: set_status_label()

FUNCTION: statusRegisterNames()

FUNCTION: statusInitWidgets()

FUNCTION: create_status_widget(rp)

parameter: RadioInfoP rp, defined in radmon.h

FUNCTION: create_status_select_button(rp)

parameter: RadioInfoP rp, defined in radmon.h

FUNCTION: getUpdateFlags(old_state, new_state)

parameter: register STATEP old_state, defined in state.h

parameter: register STATEP new_state, defined in state.h

FUNCTION: status_new_radio(rp)

parameter: RadioInfoP rp, defined in radmon.h

FUNCTION: UTMString(x, y)

parameter: double x, y, standard type

FUNCTION: unhilite_widget(w)

parameter: Widget w, defined in <X11/Intrinsic.h>

FUNCTION: status_update(rp)

parameter: RadioInfoP rp, defined in radmon.h

FUNCTION: UpdateRadioStatus()

FUNCTION: status_destroy()

FILE: timeline.c

FUNCTION: tlEnqueue(tlq, tls)

parameter: register TLQP tlq, defined in radmon.h

parameter: register TLSP tls, defined in radmon.h

FUNCTION: tlDequeue(tlq)

parameter: register TLQP tlq, defined in radmon.h

FUNCTION: tlReplenishFreeList()

FUNCTION: tlInit()

FUNCTION: tlChangeState(rp, timestamp)

parameter: RadioInfoP rp, defined in radmon.h

parameter: MSEC timestamp, defined in timeline.h

FUNCTION: tlNext(tlsp, tlep)

parameter: register TLSP *tlsp, defined in radmon.h

parameter: register TLEP *tlep, defined in radmon.h

FUNCTION: tlPrev(tlsp, tlep)

parameter: register TLSP *tlsp, defined in radmon.h

parameter: register TLEP *tlep, defined in radmon.h

FUNCTION: tlFind(rp, when, tlsp, tlep)

parameter: RadioInfoP rp, defined in radmon.h

parameter: long when, standard type

parameter: TLSP *tlsp, defined in radmon.h

parameter: TLEP *tlep, defined in radmon.h

FUNCTION: tlReadFile(f, rp)

parameter: FILE *f, defined in <stdio.h>

parameter: RadioInfoP rp, defined in radmon.h

FUNCTION: tlCountFile(rp)

parameter: RadioInfoP rp, defined in radmon.h

FUNCTION: tlWriteFile(f, rp)

parameter: FILE *f, defined in <stdio.h>

parameter: RadioInfoP rp, defined in radmon.h

FUNCTION: tlFree(rp)

parameter: RadioInfoP rp, defined in radmon.h

FILE: traffic.c

FUNCTION: TIME_TO_WINDOW_COORD(trf, t)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: MSEC t, defined in timeline.h

FUNCTION: WINDOW_TO_TIME_COORD(trf, x)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: int x, standard type

FUNCTION: TIME_TO_WINDOW_DELTA(trf, t)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: MSEC t, defined in timeline.h

FUNCTION: WINDOW_TO_TIME_DELTA(trf, x)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: int x, standard type

FUNCTION: trafficRegisterNames()

FUNCTION: trafficInitWidgets()

FUNCTION: traffic_cursor_valid(trf)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

FUNCTION: traffic_cursor_invalid(trf)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

FUNCTION: traffic_set_scrollbar(trf)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

FUNCTION: traffic_set_cursor_time(time)

parameter: MSEC time, defined in timeline.h

FUNCTION: traffic_cursor_time(p)

parameter: MSEC *p, defined in timeline.h

FUNCTION: traffic_draw_cursor(trf)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

FUNCTION: traffic_erase_cursor(trf)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

FUNCTION: traffic_erase_labels(trf, first_slot, last_slot)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: int first_slot, standard type

parameter: int last_slot, standard type

FUNCTION: traffic_erase_drawing(trf)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

FUNCTION: tune_selected(bi, tune)

parameter: BUTTON_INFOP bi, defined in traffic.h

parameter: TUNEP tune, defined in tune.h

FUNCTION: traffic_draw_label(trf, slot)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: int slot, standard type

FUNCTION: traffic_draw_slot(trf, slot, left, width)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: int slot, standard type

parameter: int left, standard type

parameter: int width, standard type

FUNCTION: traffic_draw_labels(trf, first_slot, last_slot)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: int first_slot, standard type

parameter: int last_slot, standard type

FUNCTION: traffic_format_time(trf, time)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: long time, standard type

FUNCTION: traffic_format_traffic_time(time)

parameter: MSEC time, defined in timeline.h

FUNCTION: traffic_time_to_date_field(time)

parameter: long time, standard type

FUNCTION: traffic_measure_grid(trf)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

FUNCTION: traffic_draw_grid(trf, left, width)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: int left, standard type

parameter: int width, standard type

FUNCTION: traffic_draw_slots(trf, first_slot, last_slot, left, width)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: int first_slot, standard type

parameter: int last_slot, standard type

parameter: int left, standard type

parameter: int width, standard type

FUNCTION: traffic_redraw_labels(w, trf, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: XmDrawingAreaCallbackStruct *cback, defined in <Xm/Xm.h>

FUNCTION: traffic_redraw(trf, left, top, width, height)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: int left, top, standard type

parameter: int width, height, standard type

FUNCTION: traffic_expose(w, trf, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: XmDrawingAreaCallbackStruct *cback, defined in <Xm/Xm.h>

FUNCTION: traffic_graphics_expose(w, trf, event)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: XEvent *event, defined in <X11/Xlib.h>

FUNCTION: traffic_resize(w, trf, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: XmDrawingAreaCallbackStruct *cback, defined in <Xm/Xm.h>

FUNCTION: traffic_close(w, trf, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: caddr_t cback, defined in <sys/types.h>

FUNCTION: traffic_zoom_in(w, trf, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: caddr_t cback, defined in <sys/types.h>

FUNCTION: traffic_zoom_out(w, trf, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: caddr_t cback, defined in <sys/types.h>

FUNCTION: traffic_left(trf, delta, limit)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: int delta, standard type

parameter: int limit, standard type

FUNCTION: traffic_continue_pan_left(trf, id)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: XtIntervalId id, defined in <X11/Intrinsic.h>

FUNCTION: traffic_pan_left(w, trf, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: XmAnyCallbackStruct *cback, defined in <Xm/Xm.h>

FUNCTION: traffic_right(trf, delta)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: int delta, standard type

FUNCTION: traffic_continue_pan_right(trf, id)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: XtIntervalId id, defined in <X11/Intrinsic.h>

FUNCTION: traffic_pan_right(w, trf, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: XmAnyCallbackStruct *cback, defined in <Xm/Xm.h>

FUNCTION: traffic_fit(w, trf, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: caddr_t cback, defined in <sys/types.h>

FUNCTION: traffic_freeze(w, trf, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: caddr_t cback, defined in <sys/types.h>

FUNCTION: traffic_unfreeze(w, trf, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: caddr_t cback, defined in <sys/types.h>

FUNCTION: traffic_input(w, trf, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: XmDrawingAreaCallbackStruct *cback, defined in <Xm/Xm.h>

FUNCTION: traffic_set_offset(trf, new_offset)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: int new_offset, standard type

FUNCTION: traffic_scrollbar(w, trf, cback)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: XmScrollBarCallbackStruct *cback, defined in <Xm/Xm.h>

FUNCTION: traffic_motion(w, trf, event)

parameter: Widget w, defined in <X11/Intrinsic.h>

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: XEvent *event, defined in <X11/Xlib.h>

FUNCTION: traffic_allocate_colors(trf)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

FUNCTION: traffic_create_widget(trf)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

FUNCTION: traffic_new_height(trf)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

FUNCTION: traffic_add_pos(trf, position)

parameter: TRAFFIC_INFOP trf, *defined in traffic.h*

parameter: int position, standard type

FUNCTION: traffic_delete_pos(trf, position)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: int position, standard type

FUNCTION: traffic_sub_select(w, bi, cback)

parameter: Widget *w, defined in <X11/Intrinsic.h>*

parameter: BUTTON_INFOP bi, defined in traffic.h

parameter: XmListCallbackStruct *cback, defined in <Xm/Xm.h>

FUNCTION: traffic_create_radio_button(trf, rp)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: RadioInfoP rp, defined in radmon.h

FUNCTION: traffic_create_tune_button(trf, tune)

parameter: TRAFFIC_INFOP trf, defined in traffic.h

parameter: TUNEP tune, defined in tune.h

FUNCTION: traffic_reset()

FUNCTION: traffic_new_tune(tune)

parameter: TUNEP tune, defined in tune.h

FUNCTION: traffic_new_radio(rp)

parameter: RadioInfoP rp, defined in radmon.h

FUNCTION: traffic_update()

FUNCTION: UpdateRadioTraffic()

FUNCTION: traffic_destroy()

FILE: tune.c

FUNCTION: tune_count_file()

FUNCTION: tune_write_file(f)

parameter: FILE *f, defined in <stdio.h>

FUNCTION: tune_read_file(f, version)

parameter: FILE *f, defined in <stdio.h>

parameter: int version, standard type

FUNCTION: tuneReplenishFreeList()

FUNCTION: tuneAllocate()

FUNCTION: tune_init()

FUNCTION: tune_set_name(tune1)

parameter: TUNEP tune1, defined in tune.h

FUNCTION: tune_find(mode, frequency, hopinfo)

parameter: unsigned char mode, standard type

parameter: long frequency, standard type

parameter: FHParameters *hopinfo

FUNCTION: tune_new()

FUNCTION: tune_compare(t1, t2)

parameter: register TUNEP t1, defined in tune.h

parameter: register TUNEP t2, defined in tune.h

FUNCTION: tune_destroy()

FUNCTION: tune_nth(n)

parameter: int n, standard type

FILE: version.c

No Typedefs for this file.

FILE: hTable.h

No Typedefs for this file.

FILE: radmon.h

No Typedefs for this file.

FILE: state.h

No Typedefs for this file.

FILE: style.h

No Typedefs for this file.

FILE: timeline.h

No Typedefs for this file.

FILE: traffic.h

No Typedefs for this file.

FILE: tune.h

No Typedefs for this file.