SOFTWARE DESIGN DOCUMENT
NOM CSCI (2)

June, 1991

Prepared by:

BBN Systems and Technologies,
A Division of Bolt Beranek and Newman Inc.
10 Moulton Street
Cambridge, MA 02138
(617) 873-3000 FAX: (617) 873-4315

Prepared for:

Defense Advanced Research Projects Agency (DARPA)
Information and Science Technology Office
1400 Wilson Blvd., Arlington, VA 22209-2308
(202) 694-8232, AUTOVON 224-8232

Program Manager for Training Devices (PM TRADE)
12350 Research Parkway
Orlando, FL 32826-3276
(407) 380-4518

REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1991	Software Design Document

4. TITLE AND SUBTITLE

Software Design Document NOM CSCI (2)

5. FUNDING NUMBERS

Contract Numbers:
MDA972-89-C-0060
MDA972-89-C-0061

6. AUTHOR(S)

Author not specified.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Bolt Beranek and Newman, Inc. (BBN)
Systems and Technologies; Advanced Simulation
10 Moulton Street
Cambridge, MA 02138

8. PERFORMING ORGANIZATION REPORT NUMBER

Advanced Simulation #:
9105

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency (DARPA
3701 North Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

DARPA Report Number:
None.

11. SUPPLEMENTARY NOTES

None

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Statement A: Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

Distribution Code:
A

13. ABSTRACT *(Maximum 200 words)*

A Simulation Network (SIMNET) project Software Design Document that describes the Network Operations and Management (NOM) Computer Software Configuration Item (CSCI number 2) of the SIMNET hardware and software training system for vehicle crew training and operational training.

14. SUBJECT TERMS

SIMNET Software Design Document for the NOM CSCI (CSCI 2).

15. NUMBER OF PAGES

16. PRICE CODE

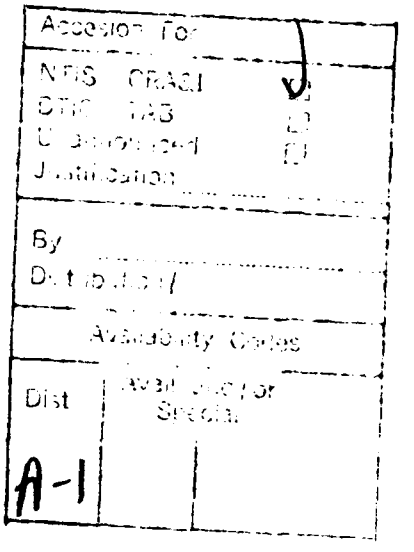
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Same as report.

SOFTWARE DESIGN DOCUMENT
NOM CSCI (2)

June, 1991

Prepared by:

BBN Systems and Technologies,
A Division of Bolt Beranek and Newman Inc.
10 Moulton Street
Cambridge, MA 02138
(617) 873-3000 FAX: (617) 873-4315



Prepared for:

Defense Advanced Research Projects Agency (DARPA)
Information and Science Technology Office
1400 Wilson Blvd., Arlington, VA 22209-2308
(202) 694-8232, AUTOVON 224-8232

Program Manager for Training Devices (PM TRADE)
12350 Research Parkway
Orlando, FL 32826-3276
(407) 380-4518

Table of Contents

1 INTRODUCTION: NETWORK OPERATIONS AND MAINTENANCE (NOM) CSCI DESCRIPTION

1.1 Background

Network Operations and Maintenance (NOM) CSCI is a system used to centralize the on-line management of SIMNET sites, with emphasis on providing tools for SIMNET site technicians to manage simulators. Technicians typically use NOM, rather than MCC (Management Command and Control), to activate simulators for inspection, maintenance, and troubleshooting. Features of NOM include simulator control, simulator status displays, and site monitoring.

Simulator control includes commands and windows that allow a technician to control individual simulators, groups of them, or all the simulators at a site from a single console. These controls allow different levels of a simulator to be activated or deactivated. A technician can issue commands to do the following activities:

- powerup or powerdown a simulator;
- shut down a simulator - which stops the UNIX operating system but does not turn off power; start or stop a simulation process; and,
- activate or deactivate a simulator into or from an exercise.

Simulator status displays include commands and windows that allow a technician to display simulator equipment status and exercise status on demand, and to display outstanding simulator events automatically. Outstanding events are those events that are pending an acknowledgement by the user interface. Event reports are propagated from event messages sent by a simulator, or from unexplained events, such as the dropout or startup of a simulator, or an equipment warning or clearing of a warning. There are also unexplained events which involve a state change without an accompanying request from NOM or MCC.

In site monitoring, NOM monitors simulator power via a power controller interface. NOM monitors equipment status and error packets via a SIMNET network interface, which NOM logs into an events log. In addition, NOM contains functions that determine which events should be sent immediately to the user interface. Such events signify a simulator state change, and will cause a color-coded alarm display.

NOM communications processes enable a technician to use the NOM console as if the technician were logging on and interacting with a SIM host terminal. This remote terminal emulation capability brings up a remote SIM window, making it possible for the technician to enter commands without having to walk to each simulator. A technician at a NOM console can also see command lines being entered by another technician at the local console of a simulator, and vice-versa. Another NOM communications feature allows a technician to view simulator output at a NOM console while the simulator is running.

1.2 External Interfaces

NOM interfaces with the following SIMNET Systems:

- Simulator (SIM) hosts;
- Management Command and Control (MCC); and,
- Power Controller

NOM external interfaces are illustrated in Figure 1.2-1, "NOM External Interfaces," and summarized in Table 1.2-1, "Summary of NOM External Interfaces."

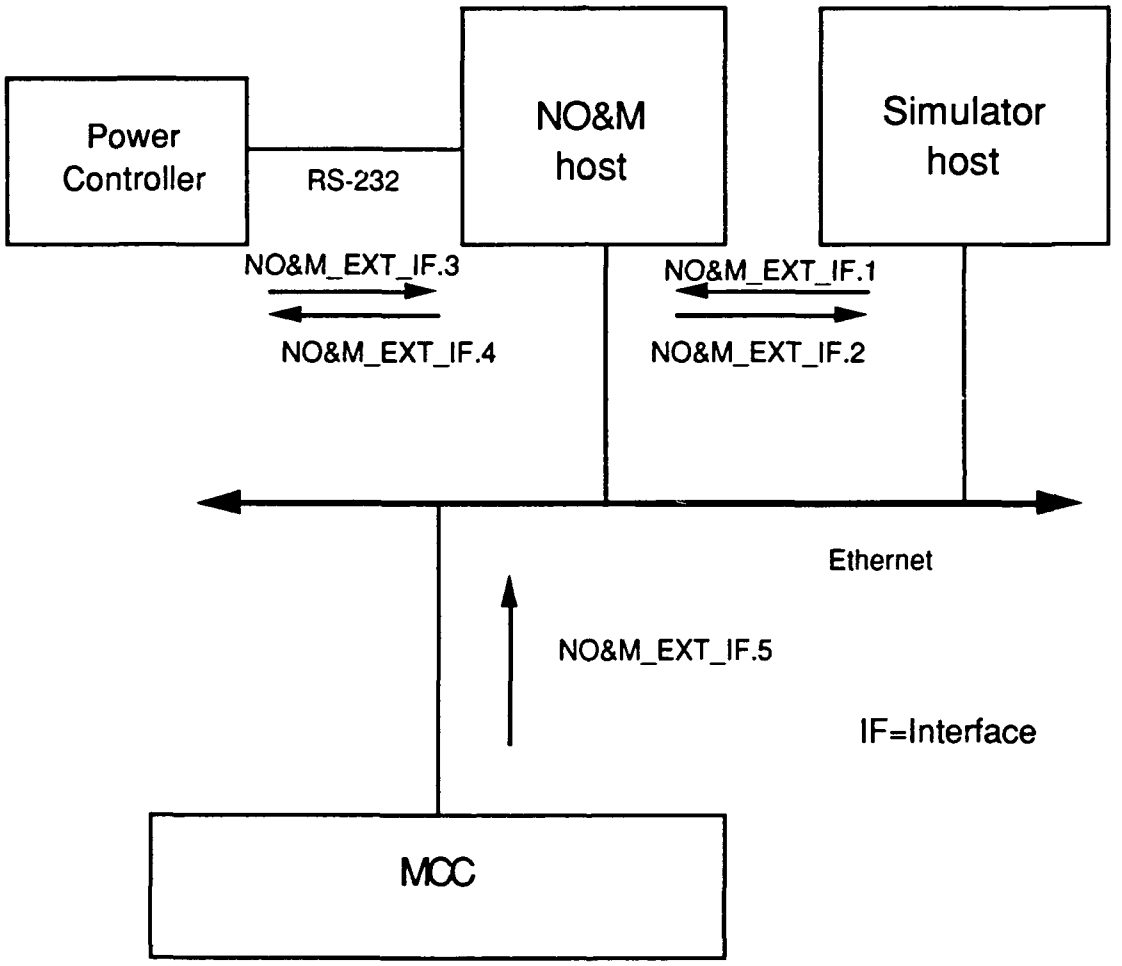


Figure 1.2-1: NOM External Interfaces.

NOM communicates with SIM hosts and MCC systems over Ethernet, and with the Power Controller using a serial RS-232 line. SIMNET simulation, management, data collection, and special NOM communications subsystem protocols are used for the Ethernet communications. A SIMNET serial protocol is used for the RS-232 connection. Table 1.2-1 summarizes the NOM External Interfaces.

Interface ID	From	To	Description	Protocol
NOM_EXT_IF1	SIM host	NOM	equipment status	Data Collection
NOM_EXT_IF1	SIM host	NOM	Error Reports	Management
NOM_EXT_IF1	SIM Host	NOM	'Heartbeat' PDU	Monitor
NOM_EXT_IF1	SIM host	NOM	User input from SIM console	Monitor
NOM_EXT_IF1	SIM host	NOM	Command output	Monitor
NOM_EXT_IF1	SIM host	NOM	Create UNIX shell	Monitor
NOM_EXT_IF1	SIM host	NOM	NSCP Process Response	Monitor
NOM_EXT_IF2	NOM	SIM host	Shell output	Monitor
NOM_EXT_IF2	NOM	SIM host	Transfer Reliability (?)	Monitor
NOM_EXT_IF2	NOM	SIM host	User input from NOM console	Monitor
NOM_EXT_IF2	NOM	SIM host	Request to Create Remote SIM UNIX shell	Monitor
NOM_EXT_IF2	NOM	SIM host	NSCP Process Request	NSCP
NOM_EXT_IF2	NOM	SIM Host	Poll Sim "Heartbeat"	Monitor.
NOM_EXT_IF2	NOM	SIM host	Request Error Reports	Monitor
NOM_EXT_IF3	Power Controller	NOM	Power Status	Serial
NOM_EXT_IF4	NOM	Power Controller	Power-related Commands	Serial
NOM_EXT_IF5	MCC	NOM	Error reports	Management

Table 1.2-1: Summary of NOM External Interfaces (refer to Figure 1.2-1, "NOM External Interfaces").

Note: 'IF'='interface'; 'NSCP'=New SIMNET Copy Protocol.

For information on Data Collection and Management protocols, refer to "SIMNET Network and Protocols," Chapters 7 and 8.

For information on SIMNET protocol data units, refer to "BBN Report No.7102--The SIMNET Network and Protocols."

1.3 Internal Structure

NOM is configured as a NOM host running NOM application software, and a communications application distributed between a set of processes on the NOM host, and a set of processes on each of the SIMNET simulator hosts. The distributed application allows a technician to manage simulators remotely, from a central location, or, to control a single simulator from its locally attached SIM host console. NOM is modular in that it can be run with less than full functionality at a site that does not have the full range of hardware.

NOM functionality may be broken down into these top-level CSCs.

- Graphical user interface (GUI);
- Management applications process (MAP);
- Interface to power controller (Lamplighter Interface);
- NOM communications;
- NOM-level processing.

The breakdown into top-level CSCs is illustrated in Figure 1.3-1, "NOM CSCI Structure." Further decomposition of top-level CSCs into lower-level CSCs and CSUs is described in Section 2.

These CSCs are implemented relatively independently within NOM as separate subsystems. Interfaces with one another are illustrated in Figure 1.3-1, "NOM Internal Interfaces," and summarized in Table 1.3-1, "Summary of NOM Internal Interfaces."

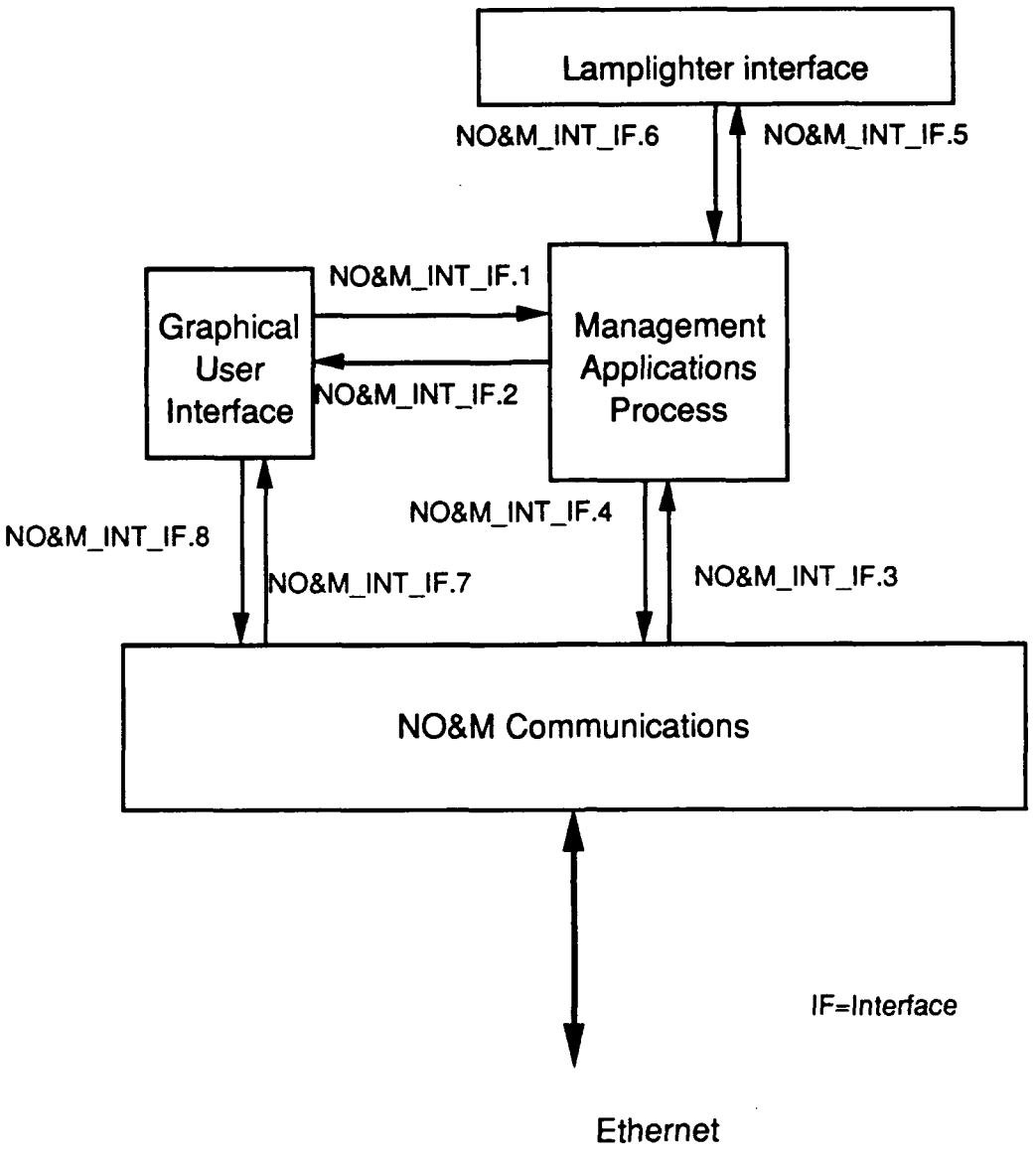


Figure 1.3-1: NOM Internal Interfaces.

Interface ID	From	To	Description	Protocol
NOM_INT_IF.1	GUI	MAP	User commands	Man.-Man.
NOM_INT_IF.1	GUI	MAP	Status Queries, including non-important events	Shared mem.
NOM_INT_IF.2	MAP	GUI	Response to user commands	Man.-Man
NOM_INT_IF.2	MAP	GUI	Important events--see dwilbert	Man.-Man.
NOM_INT_IF.3	Comm..	MAP	SIM "Heartbeat"	Monitor.
NOM_INT_IF.3	Comm.	MAP	SIM Error Reports	Monitor.
NOM_INT_IF.4	MAP	Comm.	Sim and UNIX-like Commands	Monitor.
NOM_INT_IF.4	MAP	Comm	Request Error Reports	Monitor
NOM_INT_IF.5	MAP	LL Interface	Power Requests	Man.-Man.
NOM_INT_IF.6	LL Interface	MAP	Power status	Shared Mem.
NOM_INT_IF.7	GUI	Comm.	Request for SIM UNIX shell	Monitor.
NOM_INT_IF.8	Comm.	GUI	Response to SIM UNIX shell request	Monitor

Table 1.3-1: Summary of NOM Internal Interfaces (refer to Figure 1.3-1, "NOM Internal Interfaces").

There are two kinds of communications used between NOM CSCs.

One kind of communication is based on shared memory which contains information maintained by, and accessible to, each subsystem. This more passive mechanism of inter-process communication allows other subsystems to access the information and act upon it at later times.

The NOM shared memory provides a common set of information which is accessed by all the subsystems of the NOM. All the NOM shared memory is contained in a structure called NomInfo. Each subsystem is responsible for entering its own name. NomInfo also contains the current threshold values for those equipment status values that will generate alarms. Also included are a few configuration parameters and a table which contains attributes about all the simulators.

The other kind of communication is message based, in which one process sends a message to another process (in a different CSC), which then proceeds to act on the message. This kind of communication is very active in nature; sending a message generally prompts a response. Messages in NOM are generally used to request a service of another subsystem or to alert a subsystem to an important event.

NOM CSCs communicate using IPC (Inter-Process Communications) services to set up an I/O interface. Data passed is structured according to Manager-Manager protocols or special NOM Communications protocols. CSCs share information in NOM shared memory for events that do not require immediate servicing. CSCs use special Communications IPC and data protocol when using Communications services.

Management protocol governs communications between GUI, MAP, and the Lamplighter interface. It defines the following PDUs (Protocol Data Unit):

- Event;
- Action request;
- Action response

The NOM communications CSC uses a protocol called the "monitor" protocol to control communications within the communications CSC itself, between this CSC and other NOM CSC, and between this CSC and SIM hosts. The GUI uses this protocol to open or close a session with a remote SIM software shell, and MAP uses this protocol to send commands to a SIM.

Protocol Data Units used for intra-NOM communication are summarized in Table 1.3-2, below.

PDU	CSCI/CSC From	CSCI/CSC To	Synchron-ization	Protocol	Contents
command MessageKind	rsend	fido	contingent upon AttachMes-sage	Local IPC	Execute command remote system.
command MessageKind	mon_rcmd	rover	contingent upon AttachMes-sage	Local IPC	Execute command on remote system.
attach MessageKind	mon_wind	fido	asynchronous	Local IPC	Open connection to fido.
attach messageKind	tee-in; mon_lcmd	rover	asynchronous	Local IPC	Open connection to rover.
writeMessage Kind	rsend (MAP)	fido (NOM Comms.)	contingent upon AttachMes-sage	Local IPC	Write text to remote shell.
listRequest MessageKind	rover	tee-in; mon_lmcd	contingent upon AttachMes-sage	Local IPC	
killMessage Kind	mon_wind; mon_rcmd	fido	contingent upon AttachMes-sage	Local IPC	Shut-down local connection
killMessage Kind	tee-in; mon_lcmd; mon_rcmd	rover	contingent upon AttachMes-sage	Local IPC	shut-down local connection

(Table 1.3-2 is continued on the following page.)

PDU	CSCI/CSC From	CSCI/CSC To	Synchron-ization	Protocol	Contents
monitorQuery PDUKind	fido	either fido or rover	periodic	Monitor	Opening a connection; used in polling SIMs and other NOMs; sent before sending text or command PDUs.
monitorQuery PDUKind	rover	either fido or rover		Monitor	Opening a connection; sent before sending a remote command or text.
monitorRe-sponsePDU Kind	either fido or rover	either fido or rover	Sent upon receiving monitorQuery PDU.	Monitor	Monitor process responds by identifying itself; lack of, or wrong, response closes connection.
shellOpen PDUKind	fido	rover	Contingent upon receiving MonitorRe-sponsePDU.	Monitor	Request to open a remote session
remoteCom-mandPDU Kind (See Note 1).	fido; rover	rover; fido	Contingent upon MonitorRe-sponsePDU; responds to receiving local IPC PDU, command-Message.	Monitor	Command text
shellStatus PDUKind	rover	fido	Looks for EOF at end of pseudo-tty, as well as other checking; returned in response to shellOpen PDU.	Monitor	Session Status
shellText PDUKind	fido	rover	Contingent upon receiving shellStatus PDU. In response to local IPC PDU writeMessage .	Monitor	Text for a shell.

(Table 1.3-2 is continued on the following page.)

PDU	CSCI/CSC From	CSCI/CSC To	Synchron-ization	Protocol	Contents
shellText PDUKind	rover	fido	Retransmitted unacknowled ged text up to 3 times.	Monitor	Text from a shell.
shellAckPDU Kind (See Note 2)	fido	rover	Sent upon receiving shellTextPDU	Monitor	Acknowledge shellTextPDU.
Event	MAP	GUI	Immediately sent upon failure of requested action; immediately sent when SIM state changes to alarmed state.	Manager-Manager	Action request failure; SIM alarmed state change.
Event	LL-Interface	MAP	Immediately sent upon detection of power anomaly	Manager-Manager	Power anomaly
Action Request	GUI	MAP	Asynchro-nous	Manager-Manager	Requests an Operation
Action Response (See Note 3)	MAP	GUI	acknowledge-ment of Request.with matching action ID.	Manager-Manager	Request for action has been received; action ID same as requested action ID.

Table 1.3-2: Summary of NOM Protocol Data Units (PDUs)

SIMNET management, data collection, and simulation protocols are described in "SIMNET Network and Protocols'" BBN Report No. 1702.

Some abbreviations:

'YUMM' - names a BBN library that contains monitor IPC functions.

'Pseudo-tty' - a pseudo I/O device. A pseudo tty is treated as a device for purposes of identifying input and allocating storage.

'Fido' and 'Rover' are CSUs described in Section 2.4.1.1 and Section 2.4.2.1, respectively. In accordance with modularity constraints (Section 2) they interface indirectly, through their parent CSC nodes, NOM Host Communications Processes (Section 2.4.1) and SIM Host Comms Processes (Section 2.4.2).

Notes:

1. Rover does not acknowledge messages from Fido.

2. Fido acknowledges messages from Rover. Text messages have forced sequence numbering. Character sequence numbers are used to represent positions in the text stream for the purpose of acknowledgement. If SIM fails to receive an acknowledgement for text it has sent, it will retry the transmission up to three times.

3. If the action can be performed immediately, the MAP does so and returns an actSuccess or actFailure return code. If the request will take some time, the MAP returns a response with an actAcknowledge return code.

4. Monitor PDUs are placed in Ethernet packets with IEEE 802.3 headers.

Monitor protocol parameter settings:

Number of characters by which Rover's transmissions to Fido are permitted to "get ahead" of acknowledgements received back from Fido=transmitTextBufferLength, 500 characters.

Amount of time that Rover will wait for an acknowledgement before retransmitting text to Fido = monitorRetryTimeout, 3 seconds.

Number of times Rover will transmit a particular block of characters to Fido, seeking an acknowledgement each time = maxTextTransmitAttempts, 3.

Interval with which Fido polls for any Rover processes on the network = simulatorPollInterval, 30 seconds.

Interval after which Fido declares a silent simulator to be "down" = simulatorTimeout, 90 seconds.

1.4 Configuration and Configuration Management

NOM is coded in the "C" programming language. It makes use of functions from standard "C", UNIX system calls, and libraries compiled from the public domain source of X, version 11, Release 4 (X11R4).

NOM currently runs under Concurrent Real Time Unix (RTU) version 4.0A. In most cases it will only require a recompile to operate with other versions of RTU.

The full-featured NOM configuration consists of a 5600 Concurrent host, a large capacity (380mb) disk drive, a large capacity (140mb) tape drive, Communications Machinery Corporation ethernet card, a serial control unit, and a high resolution graphics monitor.

Auxiliary files--/simnet/data

.../nom/NOM-pars:
This is an input parameter file that each site must create for configuring NOM at start-up. Entries are:

terrain <database_name>

This entry is used to set the terrain database the NOM will use during its operation.

vehicle <site> <host> <row> <column> <enet_addr> <lamp_addr>

This entry is required for each simulator icon. The site. and host.fields are the site ID and host ID assigned to a simulator. These are found in the .../assoc.def file on the simulator. Row and column identify the trailer location of the simulator; the field enet_addr is the complete ethernet address of a simulator; and the lamp_addr is the lamplighter address of the simulator in decimal.

.../nom/NOM_state:
This is a NOM output file, which stores the last values set from the "Options" command.

1.5 Terminology and Documentation

1.5.1 Terms and Abbreviations

bit map	pattern of bit settings which determine a visual display. Each "on" bit turns on a picture-element (pixel).
bitblt	bit block transfer.
client	a user's application program.
CSC	Computer Software Component. A distinct part of a computer software configuration item (CSCI). CSCs may be further decomposed into other CSCs and Computer Software Units (CSUs).
CSCI	Computer Software Configuration Item. A configuration item for computer software.
CSU	Computer Software Unit. An element of a Computer Software Component (CSC) that is separately testable.
fb	frame buffer (graphics programming).
heartbeat	indications that a host is alive and running; response PDUs sent from a SIM host to the NOM are used to indicate presence or absence of a SIM host heartbeat.
host/Host	mainframe
IF	Interface
IPC	Inter-Process Control. A protocol for local communications between NOM subsystems.
Man-Man	Manager-Manager Protocol
NOM	Network Operations and Maintenance.
PDU	Protocol Data Unit.
plane	in graphics programming, 'plane' refers to a block of raster memory for storing a bit map. One memory plane allows two bit-settings per pixel - on or off. Multiple planes provide multiple bit settings per pixel; multiple planes are used to achieve hue, layering, or depth effects.
Protocol	a set of procedures and data formats that govern data communications.

resource	in windows programming, a resource is a complex shareable set of data structures that defines a window object.
segment	in graphics programming, 'segment' refers to set of related graphics commands.
server	a program that controls the graphics system. A server acts as an intermediary between a client and system resources.
shell	a UNIX command interpreter.
SIM host	simulator host (vehicle or Management Command and Control).
Subscription	membership of a simulation entity in a multicast group; subscribing and unsubscribing are SIMNET network services provided by the SIMNET association protocol.
socket	used in communications; implemented as a message address.
Toolkit	contains sets of X-Window options, each set controlling an aspect of windows, for example, a window's geometry.
UI	User Interface
UTM	Universal terrain meridian
Widgets	X-Window "objects" such as, menus, scrollbars, and buttons.
YUMM	Part of /simnet/rcs/lib/libipc.a (Section 2.5.3.6). Contains routines for message passing.

1.5.2 References

The following documents are relevant to an understanding of the NOM and its implementation in "C":

- *The "C" Programming Language*. Kernighan and Ritchie, Prentice-Hall, 1978.
- *DOD-STD-2167A & Associated DIDs and DOD-STD-2168 & Associated DID*. David Maibor Associates, Needham Heights, MA, February 29, 1988.
- *Introducing UNIX SYSTEM V*. Rachel Morgan and Henry McGilton, New York, McGraw-Hill, 1987
- "SIMNET Data Collection and Review," BBN STC, October 1988.
- *The SIMNET Maintenance Manual*, version 6.2, March 1990.
- "SIMNET Network and Protocols," BBN Report No. 1702.

2 NOM CSC DESCRIPTIONS

In this section, top-level CSCs are decomposed into lower-level CSCs and CSUs. Each CSU is generally mapped to a source file or library. The contents of each CSU are summarized.

Some top-level CSCs are broken down into lower-level CSCs. At the end of each top-level CSC description, a breakdown into lower-level CSCs, if any, is shown, and each lower-level CSC functionally summarized. If there are no lower-level CSCs corresponding to the top-level CSC, it is decomposed into one or more CSUs. If there are lower-level CSUs, each of these, in turn, is decomposed into one or more CSUs.

Interfacing Constraints:

NOM interfaces are designed to be as modular as possible. This constrains the representation of NOM interfaces as follows:

1) Design elements at the same level of hierarchy are represented as interfacing directly only with their design peers; that is, a top-level CSC is represented as interfacing directly only with other top-level CSCs, a lower-level CSC is represented as interfacing directly with other lower-level CSCs, and a CSU, as interfacing directly only with other CSUs.

2) Design peers subordinated under different parent nodes are represented as interfacing with one another only through their parent nodes. For example, a CSU does not interface directly with a CSU belonging to a different CSC; instead, the CSUs interface through their parent CSCs.

CSU function calls represents an exception to the second constraint. One function may call any other function within its scope of linkage.

File linkage:

MAP, GUI, lamplighter_int., and NOM Comms, are linkage units. These linkage units also link to libraries under NOM-level processes (Section 2.5).

Decomposition of NOM CSCI into its top-level CSCs is illustrated in Figure 2-1.

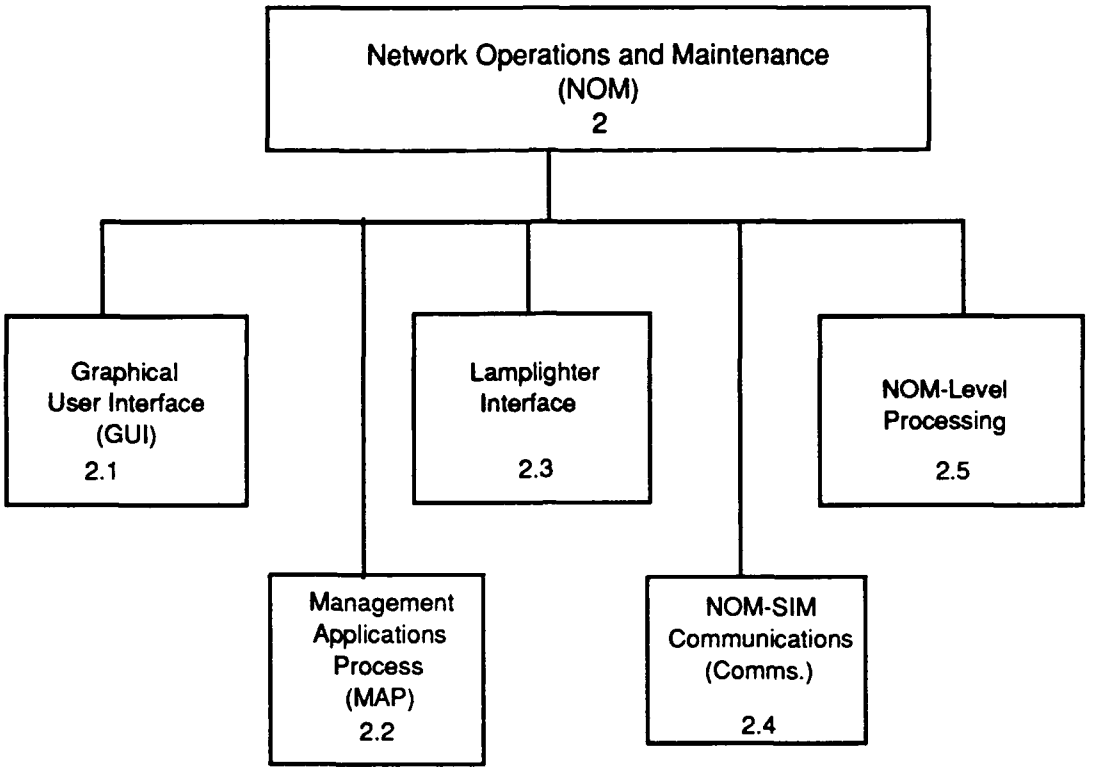


Figure 2-1: NOM CSCI Structure.

Interfaces among the NOM top-level CSCs are illustrated in Figure 1.3-2, "NOM Internal Interfaces."

2.1 Graphical User Interface (GUI) CSC Description

The graphical user interface (GUI) is a separate process in the NOM, which is used to accept commands from a human user, respond to those commands, and alert the user to asynchronous events which the NOM is monitoring.

GUI organizes the screen into three main areas: a menu panel, a drawing area, and an auxiliary window area. The drawing area provides a Mac-like interface for creating a floorplan of each site's simulator layout. The menu area provides commands for simulator controls and simulator windows for on-demand status display and remote SIM terminal login. Operations which use an auxiliary window create a scrollable vt100 simulation window. GUI uses the entire screen; however, a user has access to any commands which may be run on the particular NOM host.

GUI makes requests of the MAP using action requests. If the action can be performed immediately, the MAP does so, and returns an actSuccess or actFailure return code. If the request will take some time, the MAP returns a response with an actAcknowledge return code. If the request subsequently fails, an event is sent to the user interface.

GUI interfaces with the communications subsystem; it uses this subsystem's protocol to request interactive login capability to remote simulators, quit the open terminal session, and display command input coming from a console locally attached to a SIMNET host. GUI continuously displays information related to simulators and maintained in shared memory by the management application process (see Section 2.2, below). GUI also updates the current threshold values for equipment status values which will generate alarms.The GUI queries state information in shared memory in order to display simulator icons with the appropriate state character.

Events:

A technician can view a system events log maintained by MAP CSC (Section 2.2) and an outstanding events log for a selected simulator. Outstanding events are the ten most recent events not yet acknowledged by GUI (by selecting the Clear Events option). The outstanding events log can hold up to ten outstanding events; the eleventh (newest) event will erase event one from this log.

Events that the MAP determines to be 'important' signify a SIM state change, which causes a color-coded alarm display.

Bitmap Files

The GUI uses a number of non-executable header files which define the bit patterns of the bitmaps used in the GUI. These header files were created using a public domain bitmap editor "bitmap" (released with X11R4). However, the files are human-readable and could be edited or created by hand. These bitmap files are as follows:

m1.bitmap

This defines a generic tank icon (before the simulator is identified).

nm1.bitmap

This defines the bitmap for an M1 simulator icon.

nm2.bitmap

This defines the bitmap for an M2 simulator icon.

grid.bitmap

This defines the bitmap used to depict grid for grid-snap mode.

pencil.bitmap

This defines the pencil cursor used when naming icons.

Implementation of the GUI

The GUI uses a public domain graphics package called X. The GUI source code is linked with libraries compiled from the public domain source of X, version 11, Release 4 (X11R4). At run-time, the GUI relies upon the Masscomp X server, but the Masscomp proprietary X libraries are not used.

Some of the files used in the GUI are derived from a public domain program called "xfig". Xfig is a program provided with the source code release of X11R4. These files still contain the original headers, identifying their origin. *In GUI source files, all function names beginning with an X are from the X public domain graphics package.*

Functional breakdown of GUI into lower-level CSCs is illustrated in Figure 2.1-1.

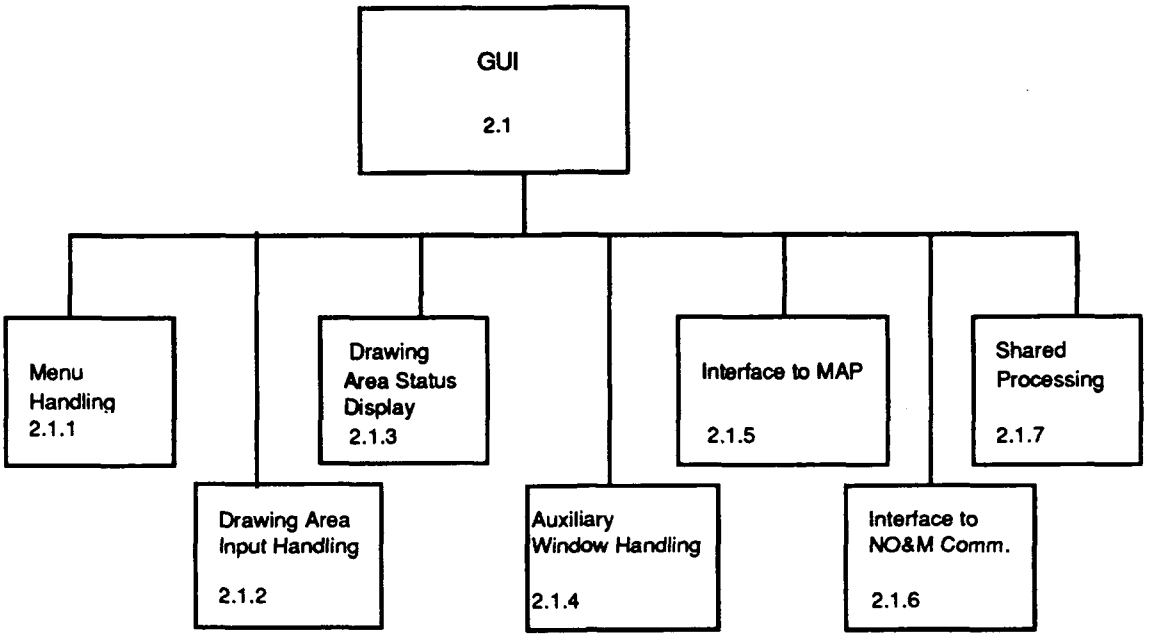


Figure 2.1-1: GUI CSC Structure

2.1.1 Menu Handling CSC Description

This lower-level CSC displays menu items and processes menu selections. Menu items include management operations, and are used to specify auxiliary window and drawing area options. Additional popup screens are sometimes used to gather additional user input.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.1.1-1.

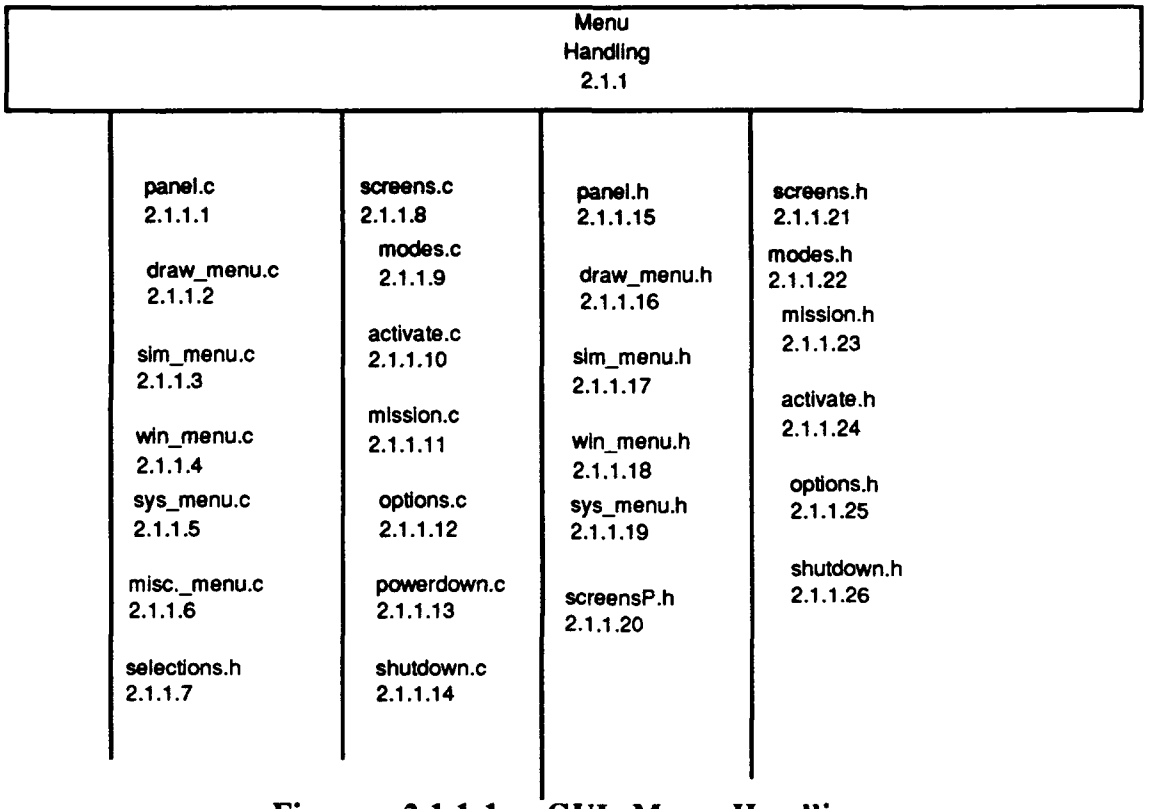


Figure 2.1.1-1: GUI--Menu Handling.

2.1.1.1 panel.c CSU Description (/simnet/cmd/nom/gui)

This CSU governs the entire control panel, which includes menus and auxiliary windows.

2.1.1.1.1 panel_InitPanel

This routine initializes the command panel of the GUI, including the mode, command, and action submenues. The label is created. The left-hand side panel, the middle panel, and the right-hand side panel are created, their widths are calculated, and their menus are placed. The routine returns 0 if successful and an error code if unsuccessful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	
Return Values		
Return Value	**Type**	**Meaning**
returnCode	ErrorCode	Unsuccessful
0	int	Successful
Calls		
Function	**Where Described**	
XtCreateWidget	public domain source of X, version 11, Release 4 (X11R4)	
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)	
XtAddActions	public domain source of X, version 11, Release 4 (X11R4)	
XtOverrideTranslations	public domain source of X, version 11, Release 4 (X11R4)	
misc_InitMenu	misc_menu.c	
draw_InitMenu	draw_menu.c	
sys_InitMenu	sys_menu.c	
win_InitMenu	win_menu.c	
sim_InitMenu	draw_menu.c	
mode_initMenu	modes.c	

Table 2.1-1: panel_InitPanel Information.

2.1.1.1.2 panel_SetupPanel

This routine finishes initializing the Panel after the environment has stabilized. The routine returns 0 if successful and an error code if unsuccessful.

Return Values		
Return Value	**Type**	**Meaning**
returnCode	ErrorCode	If 0, routine was successful; otherwise, routine was unsuccessful.
panel_ReflectCurrentSelection()	ErrorCode	If 0, routine was successful; otherwise, routine was unsuccessful.
Calls		
Function	**Where Described**	
mode_SetupMenu	modes.c	
sim_SetupMenu	draw_menu.c	
win_SetupMenu	win_menu.c	
draw_SetupMenu	draw_menu.c	
sys_SetupMenu	sys_menu.c	
misc_SetupMenu	misc_menu.c	

Table 2.1-2: panel_SetupPanel Information.

2.1.1.1.3 Panel ReflectCurrentSelection

This routine reflects the state of the currently selected simulators in the control panel, making the available selection contect-dependent, based upon the currently selected equipment. The routine returns 0 if successful and an error code if unsuccessful.

Return Values		
Return Value	**Type**	**Meaning**
returnCode	ErrorCode	unsuccessful
0	int	Successful

Calls	
Function	**Where Described**
win_ReflectCurrentSelection	win_menu.c
sim_ReflectCurrentSelection	draw_menu.c

Table 2.1-3: panel_ReflectCurrentSelection Information.

2.1.1.1.4 panel_LabelVis

This callback routine is invoked whenever the canvas gets a window exposed event from X.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
event	pointer to XVisibilityEvent	
params	pointer to String	
nparams	pointer to Cardinal	

Calls	
Function	**Where Described**
win_XtermNotify	win_menu.c
sys_XtermNotify	sys_menu.c

Table 2.1-4: panel_LabelVis Information.

2.1.1.2 draw_menu.c CSU Description (/simnet/cmd/nom/gui)

This CSU controls menu items related to the drawing area.

2.1.1.2.1 draw_GetWidth

This routine returns the maximum length (in pixels) needed to hold a state item menu entry. The number of pixels used by the longest entry are calculated and then padding is added.

Return Values		
Return Value	**Type**	**Meaning**
char_width(normalFont)*maxchars) + 2*ITEM_HORZ_PAD	int	The maximum length needed to hold a state item menu entry.
Calls		
Function	**Where Described**	
char_width		

Table 2.1-5: draw_GetWidth Information.

2.1.1.2.2 draw_InitMenu

This routine initializes the command menu in the GUI panel. A constructor is created for the entire menu. *menuArgs* is initialized with the passed constraints. The header label is created for the state menu. A widget is added for each state entry item in the menu. The fields in the state entry are initialized. The widget ID of the created menu is output into widget. The routine returns 0 if successful and an error code if unsuccessful. Parameters are represented as follows:

parent -- The parent constructor widget
spacing -- The spacing (in pixels) from the attached widget
attach -- The widget to which to attach (or align next to)
width -- The width of the menu
widget -- The output action menu widget ID

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	
spacing	int	Standard
attach	Widget	
width	int	Standard
widget	pointer to Widget	
Return Values		
Return Value	**Type**	**Meaning**
0	int	Successful
Calls		
Function	**Where Described**	
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-6: draw_InitMenu Information.

2.1.1.2.3 draw_SetupMenu

This routine initializes the popup menus.

Return Values		
Return Value	**Type**	**Meaning**
mode_PopupSetup(menuWidget)	ErrorCode	0 if successful; error code otherwise.
Calls		
Function	**Where Described**	
mode_PopupSetup	modes.c	

Table 2.1-7: draw_SetupMenu Information.

2.1.1.2.4 draw_Select

This callback routine is called when the user selects a menu item from the menu. This routine dispatches to the appropriate subprocessing.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
itemInfo	pointer to ItemInfo	draw_menu.c
Calls		
Function	**Where Described**	
mode_PopupDisplay	modes.c	
file_WriteCurrentFile	file.c	
canvas_Redisplay	canvas.c	
icon_SelectAll	icons.c	
icon_DeleteSelected	icons.c	
canvas_Revert	canvas.c	
icon_DropOn	icons.c	
icon_NameOn	icons.c	
icon_SelectOn	icons.c	
grid_Toggle	grid.c	
put_msg	msgsw.c	
DISPLAY_ERROR		

Table 2.1-8: draw_Select Information.

2.1.1.3 sim_menu.c CSU Description (/simnet/cmd/nom/gui)

This CSU creates and manages a menu block which contains commands for controlling simulators.

2.1.1.3.1 sim_GetWidth

This routine returns the maximum length (in pixels) needed to hold an item menu entry. The number of pixels used by the longest entry are calculated and then padding is added.

Return Values		
Return Value	**Type**	**Meaning**
tmp1*2*ITEM_HORZ_PAD	int	The maximum length needed to hold an item menu entry
tmp2*2*ITEM_HORZ_PAD	int	The maximum length needed to hold an item menu entry

Calls	
Function	**Where Described**
char_width	

Table 2.1-9: sim_GetWidth Information.

2.1.1.3.2 sim_InitMenu

This routine initializes the action menu in the GUI panel. A constructor is created for the entire menu. *menuArgs* is initialized with the passed constraints. The header label is created for the menu. A widget is added for each entry item in the menu. The fields in the entry are initialized. The widget ID of the created menu is output into widget. The routine returns 0 if successful and an error code if unsuccessful. The submodules are initialized. Parameters are represented as follows:

parent -- The parent constructor widget
spacing -- The spacing (in pixels) from the attached widget
attach -- The widget to which to attach (or align next to)
width -- The width of the menu
widget -- The output action menu widget ID

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	
spacing	int	Standard
attach	Widget	
width	int	Standard
widget	pointer to Widget	

Return Values		
Return Value	**Type**	**Meaning**
code	ErrorCode	Unsuccessful
0	int	Successful

Calls	
Function	**Where Described**
XtCreateManagedWidgetn	public domain source of X, version 11, Release 4 (X11R4)
XtNumber	public domain source of X, version 11, Release 4 (X11R4)
activate_Init	activate.c
powerdown_Init	powerdown.c
mission_Init	mission.c
shutdown_Init	shutdown.c

Table 2.1-10: sim_InitMenu Information.

2.1.1.3.3 sim_SetupMenu

This routine finishes intializing the menu after the rest of the environment has stabilized. The routine returns 0 if successful and an error code if unsuccessful.

Return Values		
Return Value	**Type**	**Meaning**
returnCode	ErrorCode	Unsuccessful
0	int	Successful
Calls		
Function	**Where Described**	
activate_Setup	activate.c	
powerdown_Setup	powerdown.c	
shutdown_Setup	shutdown.c	
mission_Setup	mission.c	

Table 2.1-11: sim_SetupMenu Information.

2.1.1.3.4 sim_ReflectCurrentSelection

This routine reflects the state of the currently selected equipment in the menu, making the menu context-specific. The appropiate menu entries are activated, depending upon how manu valid pieces of equipment are selected by the user. If no valid equipment pieces are selected, no commands can be performed. The number of pieces of equipmnent includes both the number of simulators and the number of icons. The routine returns 0 if successful and an error code if unsuccessful.

Return Values		
Return Value	**Type**	**Meaning**
0	int	Successful
Calls		
Function	**Where Described**	
equip_GetNumSelected	equipment.c	
XtSetValues	public domain source of X, version 11, Release 4 (X11R4)	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-12: sim_ReflectCurrentSelection Information.

2.1.1.3.5 sim_Select

This callback routine is called when the user selects a menu item from the action menu. This routine dispatches to the appropriate subprocessing. If ACTIVATE is selected for a single simulator, the user can perform a parameterized activation. Otherwise only test activations with predetermined settings are allowed. If SEL_DOWN is selected, the user must choose between emergency and normal power down.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
itemInfo	pointer to ItemInfo	
Calls		
Function	**Where Described**	
switch		
equip_GetNumSelected	equipment.c	
activate_DisplayTest	activate.c	
activate_DisplayParams	activate.c	
powerdown_DisplayChoice	powerdown.c	
shutdown_DisplayChoice	shutdown.c	
mission_DisplayChoice	mission.c	
act_SendActionToSelected	actions.c	

Table 2.1-13: sim_Select Information.

2.1.1.4 win_menu.c CSU Description (/simnet/cmd/nom/gui)

This CSU controls menu items using auxiliary vt100 windows. These windows are used for status information, error log history, and terminal passthrough.

2.1.1.4.1 win_GetWidth

This routine returns the maximum length (in pixels) needed to hold a state item menu entry. The number of pixels used by the longest entry are calculated and then padding is added.

Return Values		
Return Value	**Type**	**Meaning**
char_width(normalFont) * maxChars) + 2*ITEM_HORZ_PAD	int	The maximum length needed to hold a state item menu entry.
Calls		
Function	**Where Described**	
char_width		

Table 2.1-14: win_GetWidth Information.

2.1.1.4.2 win_InitMenu

This routine initializes the action menu in the GUI panel. A constructor is created for the entire menu. *menuArgs* is initialized with the passed constraints. The header label is created for the menu. A widget is added for each entry item in the menu. The fields in the entry are initialized. The widget ID of the created menu is output into widget. The routine returns 0 if successful and an error code if unsuccessful. The submodules are initialized. Parameters are represented as follows:

parent -- The parent constructor widget
spacing -- The spacing (in pixels) from the attached widget
attach -- The widget to which to attach (or align next to)
width -- The width of the menu
widget -- The output action menu widget ID

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	
spacing	int	Standard
attach	Widget	
width	int	Standard
widget	pointer to Widget	
Return Values		
Return Value	**Type**	**Meaning**
0	int	Successful
Calls		
Function	**Where Described**	
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	
sys_InitItems	sys_menu.c	

Table 2.1-15: win_InitMenu Information.

2.1.1.4.3 win_SetupMenu

This routine finishes initializing the menu after the rest of the environment has stabilized. This routine must be called after the initial current selection has been established. The routine returns 0 if successful and an error code if unsuccesssful.

Return Values		
Return Value	**Type**	**Meaning**
sys_SetupMenu()	ErrorCode	If 0, successful; if an error code, unsuccessful.
Calls		
Function	**Where Described**	
sys_SetupMenu	sys_menu.c	

Table 2.1-16: win_SetupMenu Information.

2.1.1.4.4 win_ReflectCurrentSelection

This routine reflects the state of the currently selected equipment in the menu, making the menu context-specific. The appropiate menu entries are activated, depending upon how manu valid pieces of equipment are selected by the user. The number of pieces of equipmnent includes both the number of simulators and the number of icons. Since all entries are currently implemented as xterms and only one is allowed at a time, the entire menu is disabled if xterm is present. Terminal passthrough can only be selected if exactly one valid piece of equipment is selected. If no valid equipment is selected, no commands can be performed. The routine returns 0 if successful and an error code if unsuccessful.

Return Values		
Return Value	**Type**	**Meaning**
0	int	Successful
Calls		
Function	**Where Described**	
equip_GetNumSelected	equipment.c	
XtSetValues	public domain source of X, version 11, Release 4 (X11R4)	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-17: win_ReflectCurrentSelection Information.

2.1.1.4.5 win_XtermNotify

This routine is called in order to reflect xterm menu items appropriately whenever an xterm is created or destroyed. Only one xterm application at a time is allowed.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
present	int	Standard
Calls		
Function	**Where Described**	
win_ReflectCurrentSelection	win_menu.c	

Table 2.1-18: win_XtermNotify Information.

2.1.1.4.6 win_ItemSelect

This callback routine is called when the user selects a menu item from the action menu. This routine dispatches to the appropriate subprocessing.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
itemInfo	pointer to ItemInfo	

Calls	
Function	**Where Described**
exec_RemoteTerminal	exec.c
equip_GetFirstSelected	equipment.c
DISPLAY_ERROR	
exec_ViewFile	exec.c

Table 2.1-19: win_ItemSelect Information.

2.1.1.5 sys_menu.c CSU Description (/simnet/cmd/nom/gui)

This CSU controls system-wide menu items.

2.1.1.5.1 sys_GetWidth

This routine returns the maximum length (in pixels) needed to hold a state item menu entry. The number of pixels used by the longest entry are calculated and then padding is added.

Return Values		
Return Value	**Type**	**Meaning**
char_width(normalFont)*maxChars + 2*ITEM_HORZ_PAD	int	The maximum length needed to hold a state item menu entry.

Calls	
Function	**Where Described**
char_width	

Table 2.1-20: sys_GetWidth Information.

2.1.1.5.2 sys_InitMenu

This routine initializes the menu in the GUI panel. The widget ID of the created menu is output into widget. A constructor is created for the entire menu. *menuArgs* is initialized with the passed constraints. The header label is created for the menu. The widget ID of the created menu is output into widget. The routine returns 0 if successful and an error code if unsuccessful. Parameters are represented as follows:

parent -- The parent constructor widget
spacing -- The spacing (in pixels) from the attached widget
attach -- The widget to which to attach (or align next to)
width -- The width of the menu
widget -- The output action menu widget ID

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	
spacing	int	Standard
attach	Widget	
width	int	Standard
widget	pointer to Widget	

Return Values		
Return Value	**Type**	**Meaning**
0	int	Successful
Calls		
Function	**Where Described**	
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	
sys_InitItems	sys_menu.c	

Table 2.1-21: sys_InitMenu Information.

2.1.1.5.3 sys_InitItems

This routine is called by **sys_InitMenu** to add a widget for each state entry item in the menu and initalize the fields in the state entry.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
menu	Widget	
width	int	Standard
Calls		
Function	**Where Described**	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-22: sys_InitItems Information.

2.1.1.5.4 sys_Setupmenu

This routine always returns 0.

Return Values		
Return Value	**Type**	**Meaning**
0	int	always returned

Table 2.1-23: sys_SetupMenu Information.

2.1.1.5.5 sys_XtermNotify

This routine is called in order to reflect xterm menu items appropriately whenever an xterm is created or destroyed. Only one xterm application at a time is allowed. The window can be closed if present. The window cannot be created if present.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
present	int	Standard

Calls	
Function	**Where Described**
XtSetValues	public domain source of X, version 11, Release 4 (X11R4)
XtNumber	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-24: sys_XtermNotify Information.

2.1.1.5.6 sys_Select

This callback routine is called when the user selects a menu item from the menu. This routine dispatches to the appropriate subprocessing.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
itemInfo	pointer to ItemInfo	sys_menu.c
Calls		
Function	**Where Described**	
exec_OpenAuxiliaryWindow	exec.c	
exec_ViewFile	exec.c	
exec_CloseAuxiliaryWindow	exec.c	
put_msg	msgsw.c	

Table 2.1-25: sys_Select Information.

2.1.1.6 misc_menu.c CSU Description (/simnet/cmd/nom/gui)

This CSU supports the Miscellaneous menu.

2.1.1.6.1 misc_InitMenu

This routine initializes the commnad menu in the GUI panel. A constructor is created for the entire menu. *menuArgs* is initialized with the passed constraints. The header label is created for the menu. A widget is added for each entry item in the menu. The fields in the entry are initialized. The widget ID of the created menu is output into widget. The routine returns 0 if successful and an error code if unsuccessful. The auxiliary screens are initialized. Parameters are represented as follows:

parent -- The parent constructor widget
spacing -- The spacing (in pixels) from the attached widget
attach -- The widget to which to attach (or align next to)
width -- The width of the menu
widget -- The output action menu widget ID

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	
spacing	int	Standard
attach	Widget	
width	int	Standard
widget	pointer to Widget	
Return Values		
Return Value	**Type**	**Meaning**
0	int	Successful
Calls		
Function	**Where Described**	
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	
opt_Init	options.c	

Table 2.1-26: misc_InitMenu Information.

2.1.1.6.2 misc_SetupMenu

This routine initializes the auxiliary screens.

Return Values		
Return Value	**Type**	**Meaning**
opt_Setup(optionsInfo.widget)	ErrorCode	0 if successful; error code if unsuccessul.
Calls		
Function	**Where Described**	
opt_Setup	options.c	

Table 2.1-27: misc_SetupMenu Information.

2.1.1.6.3 misc_Select

This callback routine is called when the user selects a menu item from the action menu. This routine dispatches to the appropriate subprocessing.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
itemInfo	pointer to ItemInfo	misc_menu.c
Calls		
Function	**Where Described**	
opt_Display	options.c	
put_msg	msgsw.c	

Table 2.1-28: misc_Select Information.

2.1.1.7 selections.h CSU Description (/simnet/cmd/nom/gui)

This CSU is included in all the menu module files (draw_menu, sim_menu, sys_menu, etc). It uniquely defines all the menu selections. This global name space was adopted to facilitate rearranging menus and putting menu items in different menus.

2.1.1.8 screens.c CSU Description (/simnet/cmd/nom/gui)

This CSU provides generic support for popup screens.

2.1.1.8.1 Initialize

This routine initializes a new screen widget. The label of the new screen widget is initialized. A form, consisting of a label and entry field next to each other, is added for each field record.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
request	Widget	
dw	ScreenWidget	

Calls	
Function	**Where Described**
XtMalloc	public domain source of X, version 11, Release 4 (X11R4)
XtCreateWidget	public domain source of X, version 11, Release 4 (X11R4)
XtNumber	public domain source of X, version 11, Release 4 (X11R4)
Max	
XtcreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)
XtManageChildren	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-29: Initialize Information.

2.1.1.8.2 ConstraintInitialize

This routine adds constraints to the initialization. The routine gets a handle to the last child of the screen (either a field or a button). If the last child is not a button, use the defaults. If the child is a button (however, not the first button), add horizontal contstrain.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
request	Widget	
new	Widget	

Calls	
Function	**Where Described**
XtIsSubclass	public domain source of X, version 11, Release 4 (X11R4)
XtIsManaged	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-30: ConstraintInitialize Information.

2.1.1.8.3 SetValues

This routine is not used in the version 6.6 release.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
current	Widget	
request	Widget	
new	Widget	

Table 2.1-31: SetValues Information.

2.1.1.8.4 XtScreenGetValueString

This routine returns the value string of the field pointer passed in *field*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	ScreenWidget	
field	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
fieldPtr->value	pointer to char	The value string of *field*

Table 2.1-32: XtScreenGetValueString Information.

2.1.1.8.5 XtScreenSetValue

This routine replaces the current value of a specified field in a screen with the new value. *widget* specifies the screen; *field* specifies the field; *newVal* specifies the new value.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	ScreenWidget	
field	int	Standard
newVal	pointer to char	Standard
Calls		
Function	**Where Described**	
XtTextReplace	public domain source of X, version 11, Release 4 (X11R4)	
XtTextSetInsertionPoint	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-33: XtScreenSetValue Infomation.

2.1.1.8.6 XtScreenAddButton

This routine adds a button to the screen specified by *screen*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
screen	Widget	
name	pointer to char	Standard
function()	pointer to function that returns void	Standard
param	caddr_t	

Calls	
Function	**Where Described**
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)
XtNumber	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-34: XtScreenAddButton Information.

2.1.1.9 modes.c CSU Description (/simnet/cmd/nom/gui)

This CSU provides a popup screen for selecting "drawing mode" Drawing modes are: creating vehicle icons; naming icons using a cursor in pencil mode; and, toggling grid-snap.

Pencil mode is supported by the parameters file, *MCC-Pars*. This file resides in the NOM, and is read by NOM when a user attempts to write a name in a SIM icon. The *Pars* file includes the following parameters: Ethernet address, SIM number, and type of vehicle, for all SIMs on site; and, the Ethernet address of the MCC hosts on site. Only valid SIM names contained in the *Pars* ile of a site can be written in a SIM icon.

The m1, grid, and pencil bitmaps are used by the modes popup screen as graphical menu items which the user may select.

2.1.1.9.1 mode_GetWidth

This routine returns the necessary width of the mode menu area. The routine assumes all icons are the same size and returns the width needed for one mode icon.

Return Values		
Return Value	**Type**	**Meaning**
items[0].icon->width	int	The width needed for one mode icon.

Table 2.1-35: mode_GetWidth Information.

2.1.1.9.2 mode_PopupSetup

This routine creates the auxiliary screen. The screen is displayed relative to the widget *parent*.. Once the widgets have been created to attach to the menu, the menu items are intialized. The "Cancel" button is added for a popup menu. mode_SetupMenu is called to complete the menu setup.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	

Return Values		
Return Value	**Type**	**Meaning**
returnCode	ErrorCode	unsuccessful
mode_SetupMenu()	ErrorCode	If 0, successful; if error code, unsuccessful

Calls	
Function	**Where Described**
XTranslateCoordinates	public domain source of X, version 11, Release 4 (X11R4)
XtWindow	public domain source of X, version 11, Release 4 (X11R4)
XDefaultRootWindow	public domain source of X, version 11, Release 4 (X11R4)
XtSetArg	public domain source of X, version 11, Release 4 (X11R4)
XtCreatePopupShell	public domain source of X, version 11, Release 4 (X11R4)
Mode_InitMenu	modes.c
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)
XtNumber	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-36: mode_PopupSetup Information.

2.1.1.9.3 mode_PopupDisplay

This routine displays the popup menu.

Calls	
Function	**Where Described**
XtPopup	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-37: mode_PopupDisplay Information.

2.1.1.9.4 mode_PopupRemove

This routine removes the popup menu.

Calls	
Function	**Where Described**
XtPopdown	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-38: mode_PopupRemove Information.

2.1.1.9.5 mode_InitMenu

This routine initializes the mode menu for selecting differnet GUI modes (i.e. grid, selection, and icon naming). The menu holder is created. The necessary padding to use up the allowed width is calculated. The header label for the state menu is created. Each of the iconic menu items is created. The routine returns 0 if successful and an error code if unsuccessful. Parameters are represented as follows:

parent -- The parent constructor widget
spacing -- The spacing (in pixels) from the attached widget
attach -- The widget to which to attach (or align next to)
width -- The wallowed width of the mode menu area
widget -- The output action menu widget ID

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	
spacing	int	Standard
attach	Widget	
width	int	Standard
widget	pointer to Widget	
Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
mode_GetWidth	modes.c	
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-39: mode_InitMenu Information.

2.1.1.9.6 mode_SetupMenu

This routine completes the initialization of the mode menu after some required external processing has been performed. The normal and inverse video pixmaps are created. Pixmaps are used for command buttons, cursors and icon diplays. The routine returns 0 if successful and an error code if unsuccessful.

Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
XtSetArg	public domain source of X, version 11, Release 4 (X11R4)	
XtGetValues	public domain source of X, version 11, Release 4 (X11R4)	
XCreatePixmapFramBitmapData	public domain source of X, version 11, Release 4 (X11R4)	
XtWindow	public domain source of X, version 11, Release 4 (X11R4)	
DefaultDepthOfScreen		
XtSetValues	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-40: mode_SetupMenu Information.

2.1.1.9.7 mode_IsActive

This routine returns whether the current mode is currently selected and active.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mode	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
items[mode].on	int	Whether the current mode is selected.

Table 2.1-41: mode_IsActive Information.

2.1.1.9.8 mode_DisplayHelp

This routine displays the passed help text as the primary mode message. This text should be refreshed after any transient error messages overwrite it temporarily.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
help	pointer to char	Standard
Calls		
Function	**Where Described**	
put_msg	msgsw.c	

Table 2.1-42: mode_DisplayHelp Information.

2.1.1.9.9 mode_RefreshMode

This routine refreshes the current mode help.

Calls	
Function	**Where Described**
put_msg	msgsw.c

Table 2.1-43: mode_RefreshMode Information.

2.1.1.9.10 mode_DisplayError

This routine displays a transient error in the display area.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
error	int	Standard
Calls		
Function	**Where Described**	
put_msg	msgsw.c	

Table 2.1-44: mode_DisplayError Information.

2.1.1.9.11 mode_ItemSelect

This routine handles the selection of an item on the specified screen.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
s	pointer to F_switch	
Calls		
Function	**Where Described**	
switch_actions		

Table 2.1-45: mode_ItemSelect Information.

2.1.1.9.12 init_switch

This routine handles the initialization of the switches.

Calls	
Function	**Where Described**
turn_on	modes.c

Table 2.1-46: init_switch Information.

2.1.1.9.13 get_grid

This routine toggles the grid on and off.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
sw	pointer to F_switch	
Calls		
Function	**Where Described**	
grid_Toggle	grid.c	

Table 2.1-47: set_grid Information.

2.1.1.9.14 turn_on

This routine turns on the passed switch.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
s	pointer to F_switch	
Calls		
Function	**Where Described**	
XtSetValues	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-48: turn_on Information.

2.1.1.9.15 turn_off

This routine turns off the passed switch.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
s	pointer to F_switch	
Calls		
Function	**Where Described**	
XtSetValues	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-49: turn_off Information.

2.1.1.9.16 switch_action

This routine handles the passed switch action.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
sw	pointer to F_switch	
Calls		
Function	**Where Described**	
turn_off	modes.c	
off_action		
turn_on	modes.c	
on_action		

Table 2.1-50: function Information.

2.1.1.10 activate.c CSU Description (/simnet/cmd/nom/gui)

This CSU provides a popup screen for setting activate options. This CSU uses an auxiliary screen to options accessable by the user for the activation of a single icon. A minimum of attributes are available since the NOM activation is intended as a "trial" activation used by technical support, rather than exercise starting activations which should be performed through the MCC. The user may activate multiple simulators at once, however, only automatically calculated values may be used for the options (which are different for each simulator). These values are calculated by the MAP in the Mini CSC.

2.1.1.10.1 activate_Init

This routine initializes the widget screen. The routine returns 0 if successful and an error code if unsuccessful.

Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	successful
code	ErrorCode	unsuccessful

Table 2.1-51: activate_Init Information.

2.1.1.10.2 activate_Setup

This routine finishes the initialization after widgets have been realized. The screen is displayed relative to the widget *parent*.. The auxiliary screen, used to set values for an activation, is created. The confirmation screen, used when multiple icons are selected for activation, is created. The routine returns 0 if successful and an error code if unsuccessful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	
Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	successful
code	ErrorCode	unsuccessful
Calls		
Function	**Where Described**	
XTranslate Coordinates	public domain source of X, version 11, Release 4 (X11R4)	
XDefault RootWindow	public domain source of X, version 11, Release 4 (X11R4)	
XtSetArg	public domain source of X, version 11, Release 4 (X11R4)	
XtCreatePopupShell	public domain source of X, version 11, Release 4 (X11R4)	
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	
XtScreenAddButton	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-52: activate_Setup Information.

2.1.1.10.3 activate_DisplayTest

This routine provides the user with an auxiliary screen, used to confirm that the user wishes to activate multiple simulators with test values.

Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	successful
code	ErrorCode	unsuccessful
Calls		
Function	**Where Described**	
XtPopup	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-53: activate_DisplayTest Information.

2.1.1.10.4 activate_DisplayParams

This routine provides the user with an Activation parameters screen filled with appropriate default values for the single selected resource. *equip* represents the equipment whose default values to display; *buf* represents a formatting buffer for field values. The message to request default activation parameter values for the passed equipment is filled in, and then the routine blocks, awaiting the default activation parameters for the currently selected resource.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
equip	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	successful
code	ErrorCode	unsuccessful
Calls		
Function	**Where Described**	
act_ActionReqPDUFill	actions.c	
DISPLAY_ERROR		
MMP_Transact	MMP.c	
XtScreenSetValue	public domain source of X, version 11, Release 4 (X11R4)	
XtPopup	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-54: activate_DisplayParams Information.

2.1.1.10.5 activate_ConfirmParam

This routine is called when the user clicks on the "Confirm" button for the activate screen. The routine resets the current NOM and GUI options to the values currently reflected in the options screen. The Activate request is based on the request used to get default values. The values returned for the paramameters query request are copied. The format of the input is checked. The location is checked to make sure that the point is actually on the terrain. Once the input has been accepted, the screen is taken down and the activation request is sent to the MA process.

Calls	
Function	**Where Described**
bcopy	
sizeof	
XtScreenGetValueString	public domain source of X, version 11, Release 4 (X11R4)
AlignStringToAlign	
DISPLAY_ERROR	
tdb_giv_utm_get_xy	libtdb in MCC CSCI
tdb_error	libtdb in MCC CSCI
tdb_p_on_database	libtdb in MCC CSCI
XtPopdown	public domain source of X, version 11, Release 4 (X11R4)
MMP_Transact	MMP.c

Table 2.1-55: activate_ConfirmParam Information.

2.1.1.10.6 activateCancelParam

This routine is called when the user clicks on the "Cancel" button for the activation screen. The routine performs any necessary cleanup.

Calls	
Function	**Where Described**
XtPopdown	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-56: activate_CancelParam Information.

2.1.1.10.7 activate_CancelTest

This routine is called when the user clicks on the "No" button for the test confirmation screen. The routine performs any necessary cleanup.

Calls	
Function	**Where Described**
XtPopdown	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-57: activate_CancelTest Information.

2.1.1.10.8 activate_ConfirmTest

This routine is called when the user clicks on the "Yes" button for the test confirmation screen. The mechanism for testing the activate all selected simulators is initiated.

Calls	
Function	**Where Described**
XtPopdown	public domain source of X, version 11, Release 4 (X11R4)
act_SendActionToSelected	actions.c

Table 2.1-58: activate_ConfirmTest Information.

2.1.1.11 mission.c CSU Description (/simnet/cmd/nom/gui)

This CSU provides a popup screen for setting simulator Mission Capable State. This file uses an auxiliary input screen to select the current mission capability status. If the state selected is not the Mission Capable state, the simulator is displayed as yellow, even if the simulator is alarmed. This prevents nonoperative simulators' alarms from distracting the NOM operator.

2.1.1.11.1 mission_Init

This routine initializes the widget screen. The routine always returns 0.

Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Always returns 0

Table 2.1-59: mission_Init Information.

2.1.1.11.2 mission_Setup

This routine completes the initialization after the widgets have been realized. The routine returns 0 if successful and an error code if unsuccessful. The screen is displayed relative to the widget *parent*. The auxiliary screen, used to select the type of mission, is created.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	
Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
XTranslateCoordinates	public domain source of X, version 11, Release 4 (X11R4)	
XDefaultRootWindow	public domain source of X, version 11, Release 4 (X11R4)	
XtSetArg	public domain source of X, version 11, Release 4 (X11R4)	
XtCreatePopupShell	public domain source of X, version 11, Release 4 (X11R4)	
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	
XtScreenAddButton	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-60: mission_Setup Information.

2.1.1.11.3 mission_DisplayChoice

This routine provides the user with an auxiliary screen used to choose the type of mission desired. The routine returns 0 if successful.

Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
XtPopup	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-61: mission_DisplayChoice Information.

2.1.1.11.4 mission_CancelChoice

This routine is called when the user clicks on the "Cancel" button for the choice of mission screen. The routine performs any necessary cleanup.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
clientData	caddr_t	
callData	caddr_t	

Calls	
Function	**Where Described**
XtPopdown	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-62: mission_CancelChoice Information.

2.1.1.11.5 mission_SetStatus

This routine is called when the user clicks on a mission status for the selected simulators. The callback param is the state to be set. *equip* is a pointer to the equipment in the nomEquipInfo table; *index* is the index of equipment in the nomEquipInfo table.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
clientData	caddr_t	
callData	caddr_t	

Calls	
Function	**Where Described**
XtPopdown	public domain source of X, version 11, Release 4 (X11R4)
icon_IsSelected	icons.c
equip_ReflectEquipState	equipment.c

Table 2.1-63: mission_SetStatus Information.

2.1.1.12 options.c CSU Description (/simnet/cmd/nom/gui)

This CSU provides a popup screen for setting NOM options (reporting thresholds). The file supports the Options auxiliary screen used to set the current operating options for the NOM and GUI. Currently, this file only sets NOM-wide threshold parameters.

2.1.1.12.1 opt_Init

This routine initializes the options screen. The routine always returns 0.

Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	successful

Table 2.1-64: opt_Init Information.

2.1.1.12.2 opt_Setup

This routine sets up the options screen after widgets have been realized. The routine returns 0 if successful and an ErrorCode if unsuccessful. The screen is displayed relative to the widget *parent*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	
Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	successful
Calls		
Function	**Where Described**	
XTranslateCoordinates	public domain source of X, version 11, Release 4 (X11R4)	
XtWindow	public domain source of X, version 11, Release 4 (X11R4)	
XDefaultRootWindow	public domain source of X, version 11, Release 4 (X11R4)	
XtSetArg	public domain source of X, version 11, Release 4 (X11R4)	
XtCreatePopupShell	public domain source of X, version 11, Release 4 (X11R4)	
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	
XtScreenAddButton	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-65: opt_Setup Information.

2.1.1.12.3 opt_Display

This routine reloads the current values for the NOM shmem table. The routine returns 0 if successful.

Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	successful
Calls		
Function	**Where Described**	
XtScreenSetValue	public domain source of X, version 11, Release 4 (X11R4)	
XtPopup	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-66: opt_Display Information.

2.1.1.12.4 opt_Confirm

This routine is called when the user clicks on the "Confirm" button for the options screen. The current NOM and GUI options are reset to the values currently reflected in the options screen. The routine returns 0 if successful.

Calls	
Function	**Where Described**
XtScreenGetValueString	public domain source of X, version 11, Release 4 (X11R4)
DISPLAY_ERROR	

Table 2.1-67: opt_Confirm Information.

2.1.1.12.5 opt_Cancel

This routine is called when the user clicks on the "Cancel" button for the options screen. The routine performs any necessary cleanup and aborts the option setting.

Calls	
Function	**Where Described**
XtPopdown	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-68: opt_Cancel Information.

2.1.1.13 powerdown.c CSU Description (/simnet/cmd/nom/gui)

This CSU handles processing related to powering down simulators. A popup screen is provided for selecting powerdown option. This auxiliary screen selects the type of powerdown or the cancellation of the operation.

2.1.1.13.1 powerdown_Init

This routine always returns 0.

2.1.1.13.2 powerdown_Setup

This routine completes the initialization after widgets have been realized. The routine returns 0 if successful and an error code if unsuccessful. The auxiliary screen used to select the type of powerdown is created. The screen is displayed relative to the widget *parent*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	
Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Successful

Calls	
Function	**Where Described**
XTranslateCoordinates	public domain source of X, version 11, Release 4 (X11R4)
XtWindow	public domain source of X, version 11, Release 4 (X11R4)
XDefaultRootWindow	public domain source of X, version 11, Release 4 (X11R4)
XtSetArg	public domain source of X, version 11, Release 4 (X11R4)
XtCreatePopupShell	public domain source of X, version 11, Release 4 (X11R4)
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)
XtNumber	public domain source of X, version 11, Release 4 (X11R4)
XtScreenAddButton	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-69: powerdown_Setup Information.

2.1.1.13.3 powerdown_DisplayChoice

This routine provides the user with an auxiliary screen used to choose the type of powerdown desired. The routine returns 0 if successful and an error code if not successful.

Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
XtPopup	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-70: powerdown_DisplayChoice Information.

2.1.1.13.4 powerdown_CancelChoice

This routine is called when the user clicks on the "Cancel" button for the choice of powerdown screen. The routine performs any necessary cleanup.

Calls	
Function	**Where Described**
XtPopdown	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-71: powerdown_CancelChoice Information.

2.1.1.13.5 powerdown_Emergency

This routine is called when the user clicks on the "Emergency" button for the choice of powerdown screen. The routine performs any necessary cleanup.

Calls	
Function	**Where Described**
XtPopdown	public domain source of X, version 11, Release 4 (X11R4)
act_SendActionToSelected	actions.c

Table 2.1-72: powerdown_Emergency Information.

2.1.1.13.6 powerdown_Normal

This routine is called when the user clicks on the "Normal" button for the choice of powerdown screen. The routine performs any necessary cleanup.

Calls	
Function	**Where Described**
XtPopdown	public domain source of X, version 11, Release 4 (X11R4)
act_SendActionToSelected	actions.c

Table 2.1-73: powerdown_Normal Information.

2.1.1.14 shutdown.c CSU Description (/simnet/cmd/nom/gui)

This CSU provides a popup screen for selecting shutdown option. The file handles the necessary processing for shutting down simulators.

2.1.1.14.1 shutdown_Init

This routine always returns 0.

2.1.1.14.2 shutdown_Setup

This routine completes the initialization after widgets have been realized. The routine returns 0 if successful and an error code if unsuccessful. The auxiliary screen used to select the type of shutdown is created. The screen is displayed relative to the widget *parent*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	
Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Successful

Calls	
Function	**Where Described**
XTranslateCoordinates	public domain source of X, version 11, Release 4 (X11R4)
XDefaultRootWindow	public domain source of X, version 11, Release 4 (X11R4)
XtSetArg	public domain source of X, version 11, Release 4 (X11R4)
XtCreatePopupShell	public domain source of X, version 11, Release 4 (X11R4)
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)
XtNumber	public domain source of X, version 11, Release 4 (X11R4)
XtScreenAddButtons	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-74: shutdown_Setup Information.

2.1.1.14.3 shutdown_DisplayChoice

This routine provides the user with an auxiliary screen used to choose the type of shutdown desired. The routine returns 0 if successful and an error code if not successful.

Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
XtPopup	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-75: shutdown_DisplayChoice Information.

2.1.1.14.4 shutdown_CancelChoice

This routine is called when the user clicks on the "Cancel" button for the choice of shutdown screen. The routine performs any necessary cleanup.

Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
XtPopdown	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-76: shutdown_CancelChoice Information.

2.1.1.13.5 shutdown_Confirm

This routine is called when the user clicks on the "Confirm" button for the choice of shutdown screen. The routine performs any necessary cleanup.

Calls	
Function	**Where Described**
XtPopdown	public domain source of X, version 11, Release 4 (X11R4)
act_SendActionToSelected	actions.c

Table 2.1-77: shutdown_Confirm Information.

2.1.1.15 panel.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for panel.c. (See 2.1.1.1.)

2.1.1.16 draw_menu.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for draw_menu.c. (See 2.1.1.2.)

2.1.1.17 sim_menu.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for sim_menu.c . (See 2.1.1 3.)

2.1.1.18 win_menu.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for win_menu.c. (See 2.1.1.4.)

2.1.1.19 sys_menu.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for sys_menu.c. (See 2.1.1.5.)

2.1.1.20 screensP.h CSU Description (/simnet/cmd/nom/gui)

This file is a private header file that includes screens.h.

2.1.1.21 screens.h CSU Description (/simnet/cmd/nom/gui)

This file is the interface file for screen.c. (See 2.1.1.8)

2.1.1.22 modes.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for modes.c. (See 2.1.1.9.)

2.1.1.23. mission.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for mission.c. (See 2.1.1.11.)

2.1.1.24 activate.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for activate.c. (See 2.1.1.10.)

2.1.1.25 options.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for options.c. (See 2.1.1.12.)

2.1.1.26 shutdown.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for shutdown.c. (See 2.1.1.14.)

2.1.2 Drawing Area Input Handling CSC Description

This lower-level CSC enables a technician to create a floor plan of a site's simulators. It includes software that supports creating, naming, positioning, as well as selection, of simulator symbols.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.1.2-1. The Drawing Area Input Handling CSC uses the m1.bitmap, nm1.bitmap, and nm2.bitmap files. Additionally, this CSU references msgsw.c and msgsw.h (Section 2.1.5.2 and Section 2.1.5.8).

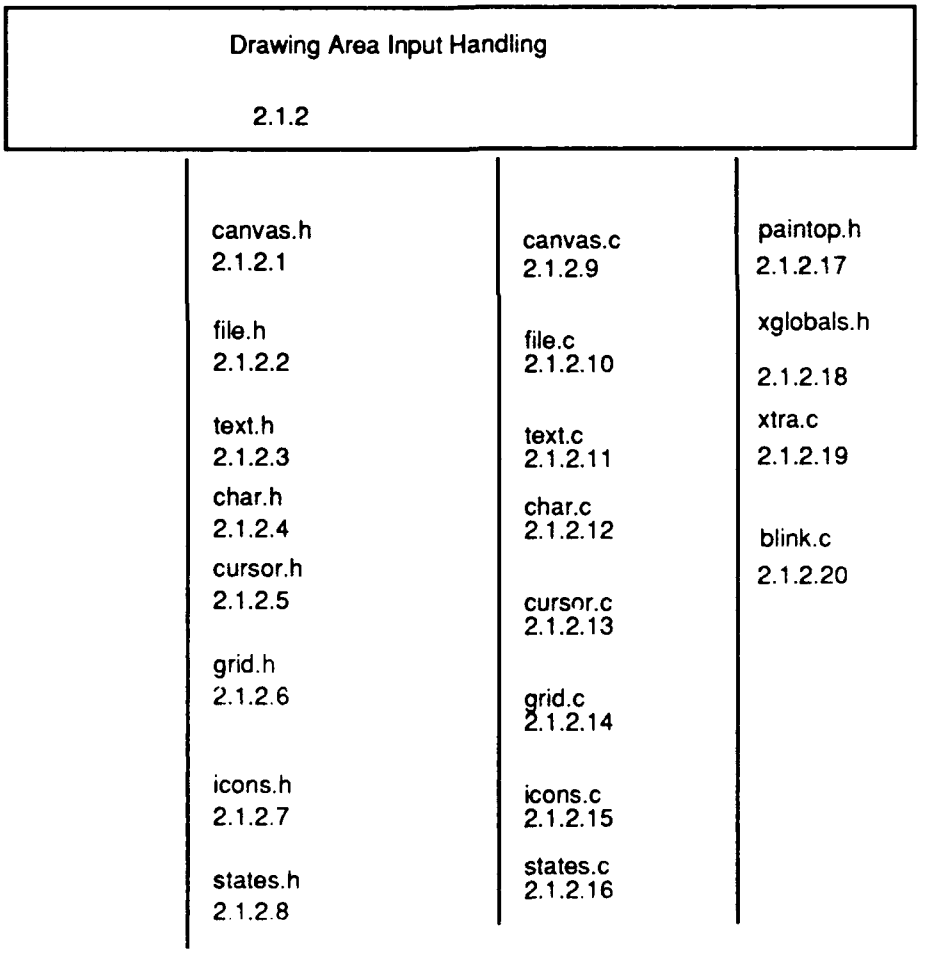


Figure 2.1.2-1: GUI--Drawing Area Input Handling.

2.1.2.1 canvas.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for canvas.c. (See 2.1.2.9.)

2.1.2.2 file.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for file.c. (See 2.1.2.10.)

2.1.2.3 text.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for text.c. (See 2.1.2.11.)

2.1.2.4 char.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for char.c. (See 2.1.2.10.)

2.1.2.5 cursor.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for cursor.c. (See 2.1.2.13.)

2.1.2.6 grid.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for grid.c. (See 2.1.2.6.)

2.1.2.7 icons.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for icons.c. (See 2.1.2.16.)

2.1.2.8 states.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for states.c. (See 2.1.2.8.)

2.1.2.9 canvas.c CSU Description (/simnet/cmd/nom/gui)

This CSU is responsible for the entire canvas area. This file supports the Macdraw-like GUI drawing area. The drawing canvas for the site depiction in the NOM GUI is supported. The user is allowed to draw a floor plan of all the simulators being managed. Note that the module icon.c provides much of the functionality of this CSU.

2.1.2.9.1 canvas_InitCanvas

This routine initializes the canvas, which is the drawing area or site depiction area of the graphich UI. The routine gets the RGB values for recolor cursor use. The routine returns 0 if successful and an error code if unsuccessful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
Return Values		
Return Value	**Type**	**Meaning**
icon_InitIcons()	ErrorCode	If 0, then successful; otherwise, unsuccessful
Calls		
Function	**Where Described**	
XtCreateWidget	public domain source of X, version 11, Release 4 (X11R4)	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	
XtGetValues	public domain source of X, version 11, Release 4 (X11R4)	
XQueryColors	public domain source of X, version 11, Release 4 (X11R4)	
XSetState	public domain source of X, version 11, Release 4 (X11R4)	
canvas_ClearCallbacks	canvas.c	
XtAddActions	public domain source of X, version 11, Release 4 (X11R4)	
XtOverrideTranslations	public domain source of X, version 11, Release 4 (X11R4)	
XtParseTranslationTable	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-78: canvas_InitCanvas Information.

2.1.2.9.2 canvas_ClearCallbacks

This routine is called to clear all callbacks which map X events to functions.

2.1.2.9.3 canvas_SetupCanvas

This routine completes the initialization of the canvas area of the GUI after the X environment and globals are established. The routine creates the drawing area and provides a handle so that xwd can dump the drawing area. The routine returns 0 if successful and an error code if unsuccessful.

Return Values		
Return Value	**Type**	**Meaning**
icon_SetupIcons	ErrorCode	If 0, then successful; otherwise, unsuccessful
Calls		
Function	**Where Described**	
XtWindow	public domain source of X, version 11, Release 4 (X11R4)	
XtStoreName	public domain source of X, version 11, Release 4 (X11R4)	
init_grid	grid.c	

Table 2.1-79: canvas_SetupCanvas Information.

2.1.2.9.4 canvas_Clear

This routine completely clears (blanks out) the drawing area.

Calls	
Function	**Where Described**
XClearArea	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-80: canvas_Clear Information.

2.1.2.9.5 canvas_Redisplay

This routine completely redisplays the contents of the canvas area.

Calls	
Function	**Where Described**
canvas_Clear	this file
icon_RedisplayAll	icons.c

Table 2.1-81: canvas_Redisplay Information.

2.1.2.9.6 canvas_Revert

This routine reverts the canvas to the initial default values.

Calls	
Function	**Where Described**
icon_DeleteAll	icons.c
file_LoadDefaultFile	file.c
icon_RedisplayAll	icons.c
panel_ReflectCurrentSelection	panel.c

Table 2.1-82: canvas_Revert Information

2.1.2.9.7 canvas_Exposed

This routine is called when the canvas receives a window exposed event from X.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
event	pointer to XExposeEvent	
params	pointer to String	
nparams	pointer to Cardinal	

Calls	
Function	**Where Described**
canvas_Redisplay	this file

Table 2.1-83: canvas_Exposed Information.

2.1.2.9.7 canvas_Left

This routine is called when the user moves the cursor out of the canvas.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
event	pointer to XCrossingEvent	
params	pointer to String	
nparams	pointer to Cardinal	

Table 2.1-84: canvas_Left Information.

2.1.2.9.8 canvas_Selected

This routine handles events in the canvas. The routine masks off irrelevant Button bits from state. The remaining bits are used for modifiers (e.g. Cntrl, Shift, Lock, Mod1-5).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
event	pointer to XButtonEvent	
params	pointer to String	
nparams	pointer to Cardinal	
Calls		
Function	**Where Described**	
XLookupString	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-85: canvas_Selected Information.

2.1.2.10 file.c CSU Description (/simnet/cmd/nom/gui)

This CSU saves and restores current canvas depiction to file.

2.1.2.10.1 file_LoadDefaultFile

This routine loads the NOM site file into the drawing area. If SAFE_FILE does not exist, a blank screen is brought up. The routine returns 0 if sucessful and an error code if unsuccessful.

Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Successful
file_LoadFile(SAVE_FILE)	ErrorCode	If 0, then successful; otherwise, unsuccessful

Calls	
Function	**Where Described**
icon_selecton	icons.c
file_LoadFile	file.c

Table 2.1-86: file_LoadDefaultFile Information.

2.1.2.10.2 file_LoadFile

This routine loads the file with the name *file* into the drawing area. *fp* is the file descriptor for the loaded file; *lineBuf* holds each line of input. The routine returns 0 if successful and an error code if unsuccessful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
file	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
errno	ErrorCode	Unsuccessful
Close(fp)	ErrorCode	If 0, then successful; otherwise, unsuccessful
Calls		
Function	**Where Described**	
icon_DeleteAll	icons.c	
icon_AddIconFromDesc	icons.c	

Table 2.1-87: file_LoadFile Information.

2.1.2.10.3 file_WriteCurrentFile

This routine saves the current site depiction into the current file. The routine returns 0 if successful.

Return Values		
Return Value	**Type**	**Meaning**
errno	ErrorCode	Unsuccessful
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
icon_AllIconsToFile	icons.c	

Table 2.1-88: file_WriteCurrentFile Information.

2.1.2.11 text.c CSU Description (/simnet/cmd/nom/gui)

This CSU supports the naming of icons.

2.1.2.11.1 prefix_length

This routine determines the character length of the prefix. "c" as part of a variable or parameter name denotes a character unit and "p" denotes a pixel unit.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
font	pointer to SFontStruct	
string	pointer to char	Standard
where_p	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
where_c	int	The character length of the prefix.
len_c	int	The character length of the prefix.
Calls		
Function	**Where Described**	
pf_textwidth		
char_advance		

Table 2.1-89: prefix_length Information.

2.1.2.12 char.c CSU Description (/simnet/cmd/nom/gui)

This CSU provides character handling routine for naming icons in the canvas area.

2.1.2.12.1 draw_cursor

This routine draws the cursor on the canvas area at the location defined by x and y.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
x	int	Standard
y	int	Standard
Calls		
Function	**Where Described**	
pw_vector		

Table 2.1-90: draw_cursor Information.

2.1.2.12.2 initialize_char_handler

This routine initializes the canvas area character handler.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
p	Window	
f	pointer to XFontStruct	
(*cr)()	pointer to function that returns int	Standard
bx	int	Standard
by	int	Standard
Calls		
Function	**Where Described**	
char_width		
char_height		
turn_on_blinking_cursor	blink.c	

Table 2.1-91: initialize_char_handler Information.

2.1.2.12.3 terminate_char_handler

This routine terminates the canvas area character handler.

Return Values		
Return Value	**Type**	**Meaning**
char_received	int	Whether the character was received
Calls		
Function	**Where Described**	
turn_off_blinkng_cursor	blink.c	

Table 2.1-92: terminate_char_handler Information.

2.1.2.12.4 erase_char_string

This routine erases a character string from the canvas area.

Calls	
Function	**Where Described**
pw_text	

Table 2.1-93: erase_char_string Information.

2.1.2.12.5 draw_char_string

This routine draws a character string on the canvas area.

Calls	
Function	**Where Described**
pw_text	
move_blinking_cursor	blink.c

Table 2.1-94: draw_char_string Information.

2.1.2.12.6 char_handler

This routine is used to enter tag names, checking on string tag name length maximum.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
c	char	Standard
Calls		
Function	**Where Described**	
erase_char_string	char.c	
draw_char_string	this file	
put_msg	msgsw.c	

Table 2.1-95: char_handler Information.

2.1.2.13 cursor.c CSU Description (/simnet/cmd/nom/gui)

This CSU creates a pool of cursors that may be grabbed dynamically by any widget.

2.1.2.13.1 setup_cursor

This routine sets up a cursor for character input.

Calls	
Function	**Where Described**
XCreateFontCursor	public domain source of X, version 11, Release 4 (X11R4)
XCreateBitmapFromData	public domain source of X, version 11, Release 4 (X11R4)
XCreatePixmapCursor	public domain source of X, version 11, Release 4 (X11R4)
XRecolorCursor	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-96: setup_cursor Information.

2.1.2.14 grid.c CSU Description (/simnet/cmd/nom/gui)

This CSU maintains canvas backdrop grid for the grid-snap option.

2.1.2.14. init_grid

This routine initializes the backdrop grid in the canvas window.

Calls	
Function	**Where Described**
XtSetArg	public domain source of X, version 11, Release 4 (X11R4)
XtGetValues	public domain source of X, version 11, Release 4 (X11R4)
XCreatePixmapFromBitmapData	public domain source of X, version 11, Release 4 (X11R4)
DefaultDepthOfScreen	
XCreatePixmapFromBitmapData	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-97: init_grid Information.

2.1.2.14.2 grid_SetValue

This routine updates the mode variable to record grid_snap mode.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
grid	int	Standard

Calls	
Function	**Where Described**
XtSetValues	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-98: grid_SetValue Information.

2.1.2.14.3 grid_Toggle

This routine toggles the grid_snap mode on and off.

Calls	
Function	**Where Described**
grid_SetValue	this file

Table 2.1-99: grid_Toggle Information.

2.1.2.15 icons.c CSU Description (/simnet/cmd/nom/gui)

This CSU supports operations on canvas icons. Icons represent simulators and other resources in the canvas.

2.1.2.15.1 icon_InitIcons

This routines zeroes out any addresses left in the NOM shared memory from the last GUI invocation. The routine returns 0 if successful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tool	Widget	
Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Successful

Table 2.1-100: icon_InitIcons Information.

2.1.2.15.2 icon_SetupIcons

This routine completes the initialization of the canvas area of the GUI after the X environmnet and globals are established. The table of colors, used to distinguish different states, is set up. The table of icons for different types is set up. For each icon state, a normal and reverse video pixmap is created for display. The routine returns 0 if successful.

Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
XCreatePixmapFromBitmapData	public domain source of X, version 11, Release 4 (X11R4)	
DefaultDepthofScreen		
makegc		

Table 2.1-101: icon_SetupIcons Information.

2.1.2.15.3 icon_GetNextSelIcon

This routine iterates through the icons in a linked list, returning the next selected icon. If NULL is passed in, the routine returns the first existing selected icon.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
icon	pointer to IconInfo	icons.c
Return Values		
Return Value	**Type**	**Meaning**
next	pointer to char	The next selected icon
0	pointer to char	The first existing selected icon

Table 2.1-102: icon_GetNextSelIcon Information.

2.1.2.15.4 icon_IsSelected

This routine returns whether the passed icon is selected or not.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
icon	pointer to IconInfo	icons.c
Return Values		
Return Value	**Type**	**Meaning**
icon->selected	int	Whether the icon is selected

Table 2.1-103: icon_IsSelected Information.

2.1.2.15.5 icon_GetEquip

This routine returns *icon*'s equipment index.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
icon	pointer to IconInfo	icons.c
Return Values		
Return Value	**Type**	**Meaning**
icon->equip	int	The equipment idex of the icon

Table 2.1-104: icon_GetEquip Information.

2.1.2.15.6 icon_DropOn

This routine is called when the user selects the icon in the panel. The routine begins dropping icons in the canvas.

Calls	
Function	**Where Described**
canvas_ClearCallbacks	canvas.c
XDefineCursor	public domain source of X, version 11, Release 4 (X11R4)
mode_DisplayHelp	modes.c

Table 2.1-105: icon_DropOn Information.

2.1.2.15.7 icon_SelectOn

This routine is called when the user deselects the current mode icon in the panel. The routine stops adding objects to the canvas. The icon selection mode is entered after cleaning up from the previous mode, and the new callbacks are set for selection mode.

Calls	
Function	**Where Described**
canvas_ClearCallbacks	canvas.c
XDefineCursor	public domain source of X, version 11, Release 4 (X11R4)
mode_DisplayHelp	modes.c

Table 2.1-106: icon_SelectOn Information.

2.1.2.15.8 icon_NameOn

This routine is called when the user selects the pencil icon. The routine names objects in the drawing area.

Calls	
Function	**Where Described**
canvas_ClearCallbacks	canvas.c
XDefineCursor	public domain source of X, version 11, Release 4 (X11R4)
mode_DisplayHelp	modes.c
char_height	
char_width	

Table 2.1-107: icon_NameOn Information.

2.1.2.15.9 icon_Draw

This routine is called to draw the passed icon in the canvas. The routine is used for initial depiction, refreshing, and selection. *icon* is the icon to draw; *selection* is used as an index which indicates whether the icon is selected; *state* is used as an index which indicates the state of the icon; *type* is the type of icon to display (i.e. M1, M2, etc.). The selection index is set at 0 or 1. The state of the icon is determined: NMCS has the highest priority for display and outstanding alarms has second highest priority. If the operating state is unkown, the state is set to unknown, otherwise, the state is set to normal. The icon is painted using the appropriate pixmaps and GCs using *selection* and *state* as indices.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
icon	pointer to IconInfo	icons.c
Calls		
Function	**Where Described**	
XCopyArea	public domain source of X, version 11, Release 4 (X11R4)	
pw_text		
ICON_NAME_X		
ICON_NAME_Y		
XDrawString	public domain source of X, version 11, Release 4 (X11R4)	
ICON_STATE_X		
ICON_STATE_Y		

Table 2.1-108: icon_Draw Information.

2.1.2.15.10 icon_AddIconToList

This routine adds the passed icon to the passed linked list of icons.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
iconList	pointer to pointer to IconInfo	icons.c
icon	pointer to IconInfo	icons.c

Table 2.1-109: icon_AddIconToList Information.

2.1.2.15.11 icon_RedisplayAll

This routine draws all icons in their current position, with the exception of any "moving icon".

Calls	
Function	**Where Described**
icon_Draw	this file

Table 2.1-110: icon_RedisplayAll Information.

2.1.2.15.13 icon_SelectAll

This routine selects all the icons in the drawing area. First, all icons are deselected, with the exception of any icon the user has clicked on. The selection state and the selection-specific menus are updated.

Calls	
Function	**Where Described**
icon_RedisplayAll	icons.c
panel_ReflectCurrentSelection	panel.c

Table 2.1-111: icon_SelectAll Information.

2.1.2.15.14 icon_WriteIconsToFile

This routine writes descriptions of the passed list of icons to the passed file. Each icon is saved in human-readable format.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
icons	pointer to IconInfo	icons.c
file	pointer to FILE	Standard

Table 2.1-112: icon_WriteIconsToFile Information.

2.1.2.15.15 icon_AllIconsToFile

This routine writes descriptions of all current icons to the passed file.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
file	pointer to FILE	Standard
Calls		
Function	**Where Described**	
icon_WriteIconsToFile	icons.c	

Table 2.1-113: icon_AllIconsToFile Information.

2.1.2.15.16 icon_AddIconFromDesc

This routine adds an icon to the global icon list, given a string description of the icon. *icon* is the icon description from the save file; *x* and *y* hold the x and y locations for each icon; *tag* holds the name tag string from the save file.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
desc	pointer to char	Standard
Calls		
Function	**Where Described**	
StringEqual	string.c	
icon_CreateIcon	icons.c	
icon_AddIconToList	icons.c	

Table 2.1-114: icon_AddIconFromDesc Information.

2.1.2.15.17 icon_DeleteAll

This routine deletes all icons from the drawing canvas and clears the canvas. *icon* is the current icon; *next* is the next icon to check; *index* is the index into the NOM shared memory table. The current selection state is fixed up for the user, including context-specific selections.

Calls	
Function	**Where Described**
icon_Delete	this file
canvas_Clear	canvas.c

Table 2.1-115: icon_DeleteAll Information.

2.1.2.15.18 icon_DeleteSelected

This routine deletes all selected objects from the drawing canvas. *icon* is the current icon; *prior* is the prior icon in the list; *next* is the next icon to check. Any icons which are selected are erased and deleted. The current selection state is fixed up for the user, including context-specific selections.

Calls	
Function	**Where Described**
icon_Erase	this file
icon_Delete	this file
panel_ReflectCurrentSelection	panel.c

Table 2.1-116: icon_DeleteSelected Information.

2.1.2.15.19 icon_Drop

This routine is called when the user tries to drop an icon in the canvas. The routine gets user's intended location for the icon, creates an unnamed icon in the appropriate position, draws the icon, and records the icon in the icon list.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
x	int	Standard
y	int	Standard

Calls	
Function	**Where Described**
canvas_ClearCallbacks	canvas.c
icon_GetHonestX	this file
icon_GetHonestY	this file
icon_CreateIcon	icons.c
put_msg	msgsw.c
icon_Draw	this file
icon_AddIconToList	icons.c

Table 2.1-117: icon_Drop Information.

2.1.2.15.20 icon_Select

This routine is called when the user clicks the right mouse button in the drawing area during icon selection. The routine selectes the clicked on icon and deselect all others. If no icon is clicked on, all icons are deselected. The selection-specific menus are updated.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
x	int	Standard
y	int	Standard
Calls		
Function	**Where Described**	
icon_Draw	this file	
icon_LocToIcon	icons.c	
panel_ReflectCurrentSelectio n	panel.c	

Table 2.1-118: icon_Select Information.

2.1.2.15.21 icon_Toggle

This routine is called when the user clicks the middle button in the drawing area. The routine inverts the state of the clicked on icon. If no icon was clicked on, the routine does nothing. The selection state and the selection-specific menus are updated.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
x	int	Standard
y	int	Standard
Calls		
Function	**Where Described**	
icon_LocToIcon	this file	
icon_Draw	this file	
panel_ReflectCurrentSelectio n	panel.c	

Table 2.1-119: icon_Toggle Information.

2.1.2.15.22 icon_NameStart

This routine is called when the user chooses to name an icon by clicking on the icon with the pencil cursor. The routine first finds the icon the user intends to name. If an icon was not clicked on, the event is ignored. Any preexisting icon name is cleared. The mapping between an existing name and the equipment table are delected. The icon drawing is refreshed. The character handling module is called to process the character I/O.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
x	int	Standard
y	int	Standard

Calls	
Function	**Where Described**
canvas_ClearCallbacks	canvas.c
icon_LocToIcon	this file
icon_Draw	this file
ICON_NAME_X	
ICON_NAME_Y	
initialize_char_handler	char.c
mode_DisplayHelp	modes.c

Table 2.1-120: icon_NameStart Information.

2.1.2.15.23 icon_NameDone

This routine is called to associate the newly entered text with the current icon as an icon name. *inputReceived* indicates whether the user input any characters; *equip* is the index of equipment in the NOM shared memory table. If there is no text, the user is assumed to be deleting an existing name. The routine checks to make sure the tag identifies a piece of equipment and that no other icon uses the identifier. The input session is terminated to make sure that input was received from the user. The name is added to the icon record structure, and the icon is associated with the correct equipment record. The resulting icon is drawn and the user is put back into the mode for selecting an icon to name. If this icon was already selected, the appropriate selection change is reflected. The current naming icon is cleared.

Calls	
Function	**Where Described**
icon_NameAbort	icons.c
icon_NameOn	this file
equip_TrailerToIndex	equipment.c
DISPLAY_ERROR	
icon_TagToIcon	this file
put_msg	msgsw.c
terminate_char_handler	char.c
StringCreateCopy	strings.c
icon_Draw	this file
panel_ReflectCurrentSelection	panel.c

Table 2.1-121: icon_NameDone Information.

2.1.2.15.24 icon_NameAbort

This routine aborts the current icon naming session. *inputReceived* indicates whether the user input any characters; *equip* is the index of equipment in the NOM shared memory table. The routine terminates the input session. If the user was updating, the update is wiped out.

Calls	
Function	**Where Described**
terminate_char_handler	char.c
icon_Draw	this file

Table 2.1-122: icon_NameAbort Information.

2.1.2.15.25 icon_MoveStart

This routine is called to move an icon when the user clicks in the canvas area during icon selection with the right mouse button. The routine determines if the user hit an icon and changes the callbacks to handle the new mode. The icon is blinked, the cursor is changed to icon shape, and the real icon is temporarily deleted in order to give the user feedback of "dragging".

Parameters		
Parameter	**Type**	**Where Typedef Declared**
x	int	Standard
y	int	Standard

Calls	
Function	**Where Described**
icon_LocToIcon	this file
canvas_ClearCallbacks	canvas.c
mode_DisplayHelp	modes.c
icon_Blink	this file
XDefineCursor	public domain source of X, version 11, Release 4 (X11R4)
icon_Erase	this file

Table 2.1-123: icon_MoveStart Information.

2.1.2.15.26 icon_MoveDone

This routine is called when the user locates an icon in the canvas to a new position. Th new position is recorded and **icon_SelectOn** is called to restore normal selection mode.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
x	int	Standard
y	int	Standard

Calls	
Function	**Where Described**
icon_GetHonestX	this file
icon_GetHonestY	this file
icon_Draw	this file
DISPLAY_ERROR	
icon_AddIconToList	icons.c
icon_Selector	this file

Table 2.1-124: icon_MoveDone Information.

2.1.2.15.27 icon_Delete

This support routine removes an icon from the icon list by removing its backward pointer from the NOM shared memory table and freeing the icon memory.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
list	pointer to pointer to IconInfo	icons.c
prior	pointer to IconInfo	icons.c
icon	pointer to IconInfo	icons.c

Table 2.1-125: icon_Delete Information.

2.1.2.15.28 icon_GetHonestX

This support routine returns the proper X coordinate for an icon to be placed. The routine accounts for grid snapping and the canvas borders. If the grid snap is on, the icon is moved to the nearest grid cross point.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
x	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
honest	int	The X coordinate for the icon to be placed.

Table 2.1-126: icon_GetHonestX Information.

2.1.2.15.29 icon_GetHonestY

This support routine returns the proper Y coordinate for an icon to be placed. The routine accounts for grid snapping and the canvas borders. If the grid snap is on, the icon is moved to the nearest grid cross point.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
y	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
honest	int	The Y coordinate for the icon to be placed

Table 2.1-127: icon_GetHonestY Information.

2.1.2.15.30 icon_Blink

This routine flashes the passed icon, leaving it in the same selection state.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
icon	pointer to IconInfo	icons.c
Calls		
Function	**Where Described**	
icon_Draw	this file	

Table 2.1-128: icon_Blink Information.

2.1.2.15.31 icon_LocToIcon

This routine returns a pointer to the first icon found at the passed location. A NULL pointer is returned if no icon is found at the location.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
x	int	Standard
y	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
0	pointer to IconInfo	no icon is found at the location
icon	pointer to IconInfo	pointer to the icon found at the passed location

Table 2.1-129: icon_IconInfo Information.

2.1.2.15.32 icon_TagToIcon

This routine returns a pointer to the first icon identified by the passed tag. A NULL pointer is returned if no icon is found in that location.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tag	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
0	pointer to IconInfo	no icon is found with that tag
icon	pointer to IconInfo	the icon identified by the tag
Calls		
Function	**Where Described**	
StringEqual	string.c	

Table 2.1-130: icon_TagToIcon Information.

2.1.2.15.33 icon_CreateIcon

This routine creates and initializes an icon with the passed values. *x* and *y* hold the location of the icon; *name* holds the name of the icon; *equip* is the index into the NOM shared memory table. The internal fields are initialized, the specified fields are set, and the backwards pointers are placed in the NOM shared memory table.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
x	int	Standard
y	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
NULL	pointer to IconInfo	memory has not been allocated for the icon
icon	pointer to IconInfo	the created icon
Calls		
Function	**Where Described**	
StringCreateCopy	string.c	

Table 2.1-131: icon_CreateIcon Information.

2.1.2.15.34 icon_Erase

This routine graphically erases the passed icon. The icon is not deleted logically. Any overlapping icons are repainted.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
icon	pointer to IconInfo	icons.c
Calls		
Function	**Where Described**	
XClearArea	public domain source of X, version 11, Release 4 (X11R4)	
icon_Draw	this file	

Table 2.1-132: icon_Erase Information.

2.1.2.16 states.c CSU Description (/simnet/cmd/nom/gui)

This CSU provides simulator state character for each icon.

2.1.2.16.1 state_GetWidth

This routine returns the maximum length (in pixels) needed to hold a state item menu entry. The number of pixels used by the longest entry are calculated and then padding is added.

Return Values		
Return Value	**Type**	**Meaning**
char_width(normalFont) * maxChars) + 2*ITEM_HORZ_PAD	int	The maximum length needed to hold a state item menu entry.
Calls		
Function	**Where Described**	
char_width		

Table 2.1-133: state_GetWidth Information.

2.1.2.16.2 state_InitMenu

This routine initializes the action menu in the GUI panel. A constructor is created for the entire menu. *menuArgs* is initialized with the passed constraints. The header label is created for the menu. A widget is added for each entry item in the menu. The fields in the entry are initialized. The widget ID of the created menu is output into widget. The routine returns 0 if successful and an error code if unsuccessful. Parameters are represented as follows:

parent -- The parent constructor widget
spacing -- The spacing (in pixels) from the attached widget
attach -- The widget to which to attach (or align next to)
width -- The width of the menu
widget -- The output action menu widget ID

Parameters		
Parameter	**Type**	**Where Typedef Declared**
parent	Widget	
spacing	int	Standard
attach	Widget	
width	int	Standard
widget	pointer to Widget	
Return Values		
Return Value	**Type**	**Meaning**
0	int	Successful
Calls		
Function	**Where Described**	
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-134: win_InitMenu Information.

2.1.2.16.3 state_SetupMenu

This routine completes the initialization of the menu after the rest of the environment has stabilized. This routine must be called after the initial current selection has been established. The routine returns 0 if successful and an error code if unsuccesssful.

Return Values		
Return Value	**Type**	**Meaning**
state_ReflectCurrentSelection()	ErrorCode	If 0, then successful; otherwise, unsuccessful
returnCode	ErrorCode	If 0, then successful; otherwise, unsuccessful
Calls		
Function	**Where Described**	
activate_Setup	activate.c	
state_ReflectCurrentSelection	states.c	

Table 2.1-135: win_SetupMenu Information.

2.1.2.16.4 state_ReflectCurrentSelection

This routine reflects the state of the currently selected equipment in the menu, making the menu context-specific. One selected simulator for the states menu is handled by this routine. Information for the passed simulator's state is found. The menu and the specific entries that are reachable from the new current state are activated. The X argument values for each item in the state menu are set up based on the current state of the passed equipment. The routine returns 0 if successful and an error code if unsuccessful.

Return Values		
Return Value	**Type**	**Meaning**
ERROR_INTERNAL	ErrorCode	Unsuccessful
0	ErrorCode	Successful

Calls	
Function	**Where Described**
icon_GetNextSelIcon	icons.c
XtSetValues	public domain source of X, version 11, Release 4 (X11R4)
XtNumber	public domain source of X, version 11, Release 4 (X11R4)
icon_GetEquip	icons.c
state_StateIDToStateInfo	this file

Table 2.1-136: win_ReflectCurrentSelection Information.

2.1.2.16.5 state_StateSelect

This callback routine is called when the user selects a menu item from the action menu. This routine dispatches to the appropriate subprocessing. Only one selected menu context is handled for the states menu. An activation request is sent for the selected state and resource.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
widget	Widget	
itemInfo	pointer to StateItemInfo	win_menu.c

Calls	
Function	**Where Described**
icon_GetNextSelIcon	icons.c
icon_GetEquip	icons.c
DISPLAY_ERROR	
equip_ReflectEquipState	equipment.c
activate_DisplayParams	activate.c
act_ActionReqPDUFill	actions.c
MMP_SendRequest	MMP.c

Table 2.1-137: win_ItemSelect Information.

2.1.2.16.6 state_StateIDToStateInfo

This routine looks up the StateItemInfo record for the passed state ID and returns a pointer to it. Zero is returned if the lookup is successful, and a NULL pointer is returned if the lookup is unsuccessful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
stateID	int	Standard

Return Values		
Return Value	**Type**	**Meaning**
stateItems[count]	pointer to StateItemInfo	the pointer to the StateItemInfo record for the passed state ID
0	pointer to StateItemInfo	unsuccessful

Table 2.1-138: state_StateIDToStateInfo Information.

2.1.2.16.7 state_GetEquipStateStr

This routine gets the string representing the state of the passed equipment.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
equip	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
itemInfo->label	pointer to char	state of equip
""	pointer to char	unsuccessful
Calls		
Function	**Where Described**	
state_StateIDToStateInfo	this file	

Table 2.1-139: state_GetEquipStateStr Information.

2.1.2.17 paintop.h CSU Description (/simnet/cmd/nom/gui)

This CSU contains definitions for screen painting.

2.1.2.18 xglobals.h CSU Description (/simnet/cmd/nom/gui)

This CSU includes X Windows toolkit and contains supporting global variables and definitions.

2.1.2.19 xtra.c CSU Description (/simnet/cmd/nom/gui)

This CSU provides original support for converting "fig" programs to X.

The following utility routines are defined:

pf_textwidth()
makegc()
init_gc()
CvtStringToFloat()
CvtIntToFloat()
fix_converters()

2.1.2.20 blink.c CSU Description (/simnet/cmd/nom/gui)

This CSU causes character input cursor to blink.

2.1.2.20.1 turn_on_blinking_cursor

This routine causes the input cursor to start blinking.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
draw_cursor()	pointer to function returning int	Standard
erase_cursor	pointer to function returning int	Standard
x	int	Standard
y	int	Standard
see	long	Standard
usee	long	Standard

Calls	
Function	**Where Described**
draw_Cursor	char.c
erase_Cursor	
notify_set_itimer_func	
setitimer	

Table 2.1-140: turn_on_blinking_cursor Information.

2.1.2.20.2 turn_off_blinking_cursor

This routine causes the input cursor to stop blinking.

Calls	
Function	**Where Described**
setitimer	
notify_set_itimer_func	

Table 2.1-141: turn_off_blinking_cursor Information.

2.1.2.20.3 blink

This routine causes the cursor to blink and returns 0 if successful.

Return Values		
Return Value	**Type**	**Meaning**
0	int	Successful

Table 2.1-142: blink Information.

2.1.2.20.4 move_blinking_cursor

This routine moves the blinking cursor to the passed location.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
x	int	Standard
y	int	Standard

Table 2.1-143: move_blinking_cursor Information.

2.1.3 Drawing Area Status Display CSC Description

This lower-level CSC supports both color-coded alarmed status display of simulator symbols and letter-coded SIM states display within a simulator symbol. This CSC also supports a documentation line containing help or error messages. The letter codes are as follows:

P the simulator is powered down; it has no power.

S the simulator is shut down; it has power but the UNIX operating system is not running.

I The simulator is idle; UNIX is booted, but the simulation process is not running.

R The simulator is running a simulation process, but has not been activated into an exercise.

A The simulator is active; it is part of an exercise.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.1.3-1.

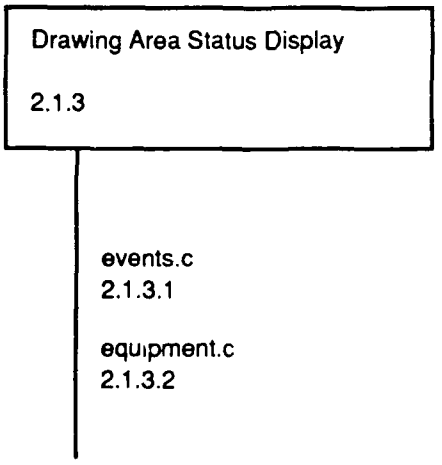


Figure 2.1.3-1: GUI--Drawing Area Status Display.

2.1.3.1 events.c CSU Description (/simnet/cmd/nom/gui)

This CSU processes incoming MMP event notifications.

2.1.3.1.1 event_EventDisplay

This routine displays an event to the user.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
eventPDU	pointer to MMP_PDUStruct	

Table 2.1-144: event_EventDisplay Information.

2.1.3.1.2 event_EventDispatch

This routine processes the receipt of an event.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
eventPDU	pointer to MMP_PDUStruct	
Calls		
Function	**Where Described**	
event_EventDisplay	events.c	
equip_ReflectEquipState	equipment.c	

Table 2.1-145: event_EventDispatch Information.

2.1.3.2 equipment.c CSU Description (/simnet/cmd/nom/gui)

This CSU provides information about simulator equipment (hosts). The file maintains and accesses information from the NOM equipment table. Routines in this file access information about equipment, where equipment is represented by an integer index into the NOM shared memory table.

2.1.3.2.1 equip_TrailerToIndex

This routine is used to get the index of a simulator matching the passed trailer and trailer element. The routine returns the simulator index if successful and -1 if the simulator is not found, or if NULL pointer is passed in.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trailerloc	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
-1	int	simulator not found or NULL pointer passed
index	int	index of simulator

Calls	
Function	**Where Described**
equip_StringMatchesTrailer	equipment.c

Table 2.1-146: equip_TrailerToIndex Information.

2.1.3.2.2 equip_StringMatchesTrailer

This routine returns TRUE if the passed string represents the passed trailer location. *trailer* is the trailer number; *element* is the position within the trailer.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
string	pointer to char	Standard
match	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	string does not represent the trailer loction
TRUE	int	string represents the trailer location

Table 2.1-147: equip_StringMatchesTrailer Information.

2.1.3.2.3 equip_ReflectEquipState

This routine refreshes the context-specific menus reflecting the state of the passed equipment.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
equip	long	Standard
Calls		
Function	**Where Described**	
icon_Draw	icons.c	
icon_IsSelected	icons.c	
panel_ReflectCurrentSelection	panel.c	

Table 2.1-148: equip_ReflectEquipState Information.

2.1.3.2.4 equip_GetNumSelected

This routine returns the number of currently selected pieces of equipment, ignoring selected unnamed icons. *count* is the current count of selected equipment.

Return Values		
Return Value	**Type**	**Meaning**
count	int	the number of selected pieces of equipment
Calls		
Function	**Where Described**	
icon_GetNextSelIcon	icons.c	
icon_GetEquip	icons.c	

Table 2.1-149: equip_GetNumSelected Information.

2.1.3.2.5 equip_GetFirstSelected

This routine returns the index of the first encountered selected equipment entry in the equipInfoTable. If no equipment entry is selected, the routine returns -1.

Return Values		
Return Value	**Type**	**Meaning**
index	int	the index of the first selected equipment entry
-1	int	no equipment entry selected
Calls		
Function	**Where Described**	
icon_IsSelected	icons.c	

Table 2.1-150: equip_GetFirstSelected Information.

2.1.4 Auxiliary Window Handling CSC Description

This lower-level CSC enables these auxiliary windows to be generated.

Console Terminal, used as a remote terminal window on a selected simulator.

Program Terminal, used to start/stop a simulator process and view its output.

Equipment Status, used for viewing equipment status of selected simulators.

Exercise Status, used for viewing exercise status of selected simulators.

Outstanding Events, used for viewing Outstanding Events log.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.1.4-1. Additionally, this CSC uses the panel.c CSU (Section 2.1.1.1)

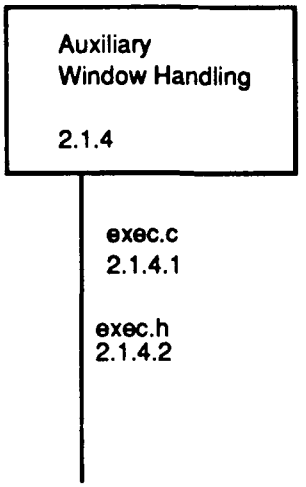


Figure 2.1.4-1 GUI--Auxiliary Window Handling

2.1.4.1 exec.c CSU Description (/simnet/cmd/nom/gui)

This CSU provides procedures for spawning subprocesses.

2.1.4.1.1 exec_Command

This routine provides fast execution of child. The first item in command must be the complete pathname of a program. The routine returns the pid of child if successful and 0 if unsuccessful. The passed command string *cmd* is broken into a program name and argument vector. *prog* is a pointer to the file to be executed; *argv* are the arguments to the program; *pid* is a process ID returned by **vfork**().

Parameters		
Parameter	**Type**	**Where Typedef Declared**
cmd	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
pid	int	pid of the child
0	int	unsuccessful

Table 2.1-151: exec_Command Information.

2.1.4.1.2 exec_OpenAuxiliaryWindow

This routine is used to execute any child processes which create auxiliary windows by executing the process which creates the auxiliary window.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
cmd	pointer to char	Standard

Calls	
Function	**Where Described**
exec_command	this file

Table 2.1-152: exec_OpenAuxiliaryWindow Information.

2.1.4.1.3 exec_CloseAuxiliaryWindow

This routine closes the current auxiliary window.

2.1.4.1.4 exec_RemoteTerminal

This routine is used to execute an auxiliary window containing a remote terminal to the specified machine using the specified channel. The machine is specified as the machine's index into the NOM shared memory table.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
machine	int	Standard
char	short	Standard
Calls		
Function	**Where Described**	
exec_OpenAuxiliaryWindow	this file	

Table 2.1-153: exec_RemoteTerminal Information.

2.1.4.1.5 exec_ViewFile

This routine is executes an auxiliary window containing a "tail" of the specified file. The -e option executes the tail command within it. When the tail is complete, the shell automatically dies. The tail command used -160f as an argument to print the last 160 lines of a specified file. The 'f' signifies to print out any newly appended lines of the file while the tail is running.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
file	pointer to char	Standrad
Calls		
Function	**Where Described**	
exec_OpenAuxiliaryWindow	this file	

Table 2.1-154: exec_ViewFile Information.

2.1.4.2 exec.h CSU Description (/simnet/cmd/nom/gui)

This is the interface file for exec.c.

2.1.5 Interface to MAP CSC Description

This lower-level CSC formats and sends command action requests to MAP and receives SIM event information and error reports from MAP. This CSC accesses a shared memory table for SIM status information and uses Manager-Manager protocol to communicate with MAP.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.1.5-1. Additionally, this CSC uses the CSUs, events.c, equipment.c (Section 2.1.4.1, Section 2.1.4.20).

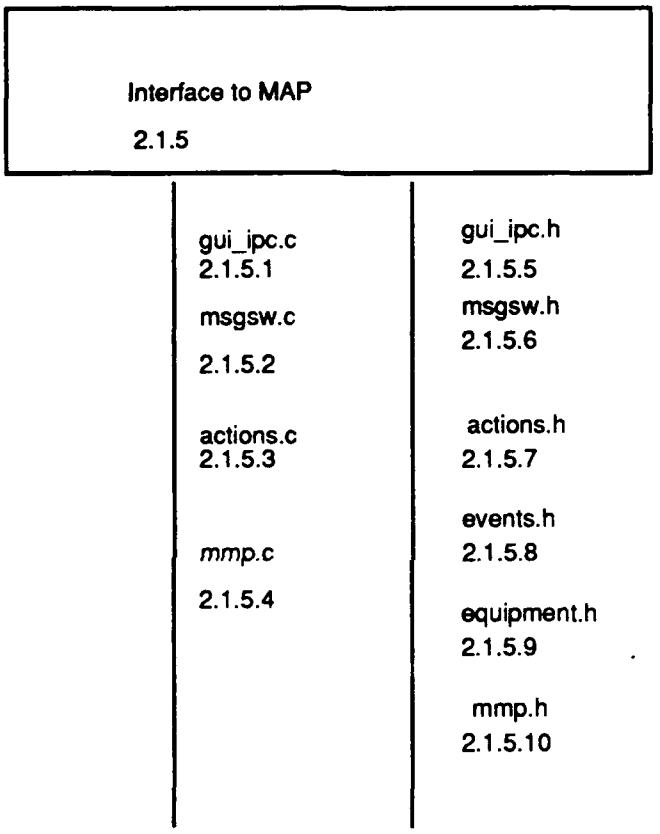


Figure 2.1.5-1 GUI--Interface to MAP

Uses of GUI interfaces with MAP:

The appropriate menu handling code uses supporting routines in actions.c. to initiate action requests. In turn, actions.c uses supporting routines in mmp.c to format and send Manager-Manager requests. Mmp.c uses supporting routines in gui_ipc.c to communicate with the MAP process.

If the GUI receives an event pertaining to a depicted simulator, the drawing of the corresponding icon may be updated. Thus, an incoming event may cause a simulator's icon to change color or cause the status character to change. The current status and color of each icon is determined by routines in equipment.c.

2.1.5.1 gui_ipc.c CSU Description (simnet /cmd/nom/gui)

This CSU provides high-level interface IPC for the GUI process using the BBN-developed "YUMM" library.

2.1.5.1.1 IPC_InitIPC

This routine initializes the IPC for the NOM graphical UI. The NOM shared memory structure, used during the rest of initialization., is attached and initialized.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
procName	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
0	ErrorCode	Successful

Table 2.1-155: IPC_InitIPC Information.

2.1.5.1.2 IPC_SetupIPC

This routine opens communications with the server process. The YUMM usage is initialized. The NOM is restricted to one GUI running at a time. If a socket has this process name associated with it, it must be deleted (the process probably died without closing). The message handler is registered to receive messages. The routine returns 0 if successful and an error code if unsuccessful.

Return Values		
Return Value	**Type**	**Meaning**
ERROR_YUMM_IPC	ErrorCode	Unsuccessful
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
LkUpSocket	libipc in MCC CSCI	
CloseSocket	libipc in MCC CSCI	
OpenSocket	libipc in MCC CSCI	

Table 2.1-156: function Information.

2.1.5.1.3 IPC_ProcessMessage

This routine is the YUMM message receipt handler for any protocol type. The routine is invoked through the YUMM callback mechanism. Alarms are disabled while messages are being processed, then reenabled when finished. The message is dispatched to the proper protocol handler.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	int	Standard
length	int	Standard
kind	long	Standard
info	long	Standard
data	pointer to register char	Standard
Calls		
Function	**Where Described**	
MMP_ProcessPDU	MMP.c	
AlarmsEnabled	libipc in MCC CSCI	

Table 2.1-157: IPC_ProcessMessage Information.

2.1.5.1.4 IPC_TraceMessage

This routine is the handler used when tracing YUMM messages. The routine is invoked through the YUMM callback mechanism. The routine assumes the message is a NOM MMP PDU.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	char	Standard
tid	char	Standard
dstSkt	YUMMSocket	libipc
rspSkt	YUMMSocket	libipc
kind	long	Standard
length	int	Standard
data	pointer to char	Standard
Calls		
Function	**Where Described**	
MMP_TracePDU	MMP.c	

Table 2.1-158: IPC_TraceMessage Information.

2.1.5.1.5 IPC_SendMessage

This routine sends data to the specified process using YUMM IPC. Each time a message is sent, a YUMM lookup is performed in order to find any changes in the sockets. The routine returns 0 if successful and an error code if unsuccessful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
destination	pointer to char	Stnadard
kind	long	Standard
length	int	Standard
data	pointer to char	Standard

Return Values		
Return Value	**Type**	**Meaning**
ERROR_NO_DEST	ErrorCode	Unsuccessful
ERROR_YUMM	ErrorCode	Unsuccessful
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
LkUpSocket	libipc in MCC CSCI	
SendMsg	libipc in MCC CSCI	

Table 2.1-159: IPC_SendMessage Information.

2.1.5.1.6 IPC_SendRequest

This routine sends data to the specified process using YUMM IPC. A reply is expected, however the routine does not Block. Each time a message is sent, a YUMM lookup is performed in order to find any changes in the sockets. The routine returns 0 if successful and an error code if unsuccessful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
dest	pointer to char	Standard
kind	long	Standard
length	int	Standard
data	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
ERROR_NO_DEST	ErrorCode	Unsuccessful
ERROR_YUMM	ErrorCode	Unsuccessful
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
LkUpSocket	libipc in MCC CSCI	
SendReq		

Table 2.1-160: IPC_SendRequest Information.

2.1.5.1.7 IPC_Transact

This routine sends data to the specified process using YUMM IPC. The routine Blocks, waiting for a response to the message. Each time a message is sent, a YUMM lookup is performed in order to find any changes in the sockets.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
dest	pointer to char	Standard
kind	long	Standard
length	int	Standard
data	pointer to char	Standard
rspkind	pointer to int	Standard
rspLen	pointer to int	Standard
rspData	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
ERROR_NO_DEST	ErrorCode	Unsuccessful
ERROR_YUMM	ErrorCode	Unsuccessful
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
LkUpSocket	libipc in MCC CSCI	
Transact	libipc in MCC CSCI	

Table 2.1-161: IPC_Transact Information.

2.1.5.1.8 IPC_CleanUp

This routine cleans up the IPC resources.

Calls	
Function	**Where Described**
YUMMCleanUp	libipc in MCC CSCI

Table 2.1-162: IPC_CleanUp Information.

2.1.5.2 msgsw.c CSU Description (/simnet/cmd/nom/gui)

This CSU displays errors which are reported by the MAP in the message bar below the canvas. Errors occur when the GUI can't get the message to the MAP or if the MAP itself reports an error.

This CSU also displays help for current canvas operation.

2.1.5.2.1 init_msg

This routine initializes the message bar. The routine returns 1 if successful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tool	Widget	
Return Values		
Return Value	**Type**	**Meaning**
1	int	Successful
Calls		
Function	**Where Described**	
XtCreateManagedWidget	public domain source of X, version 11, Release 4 (X11R4)	
XtNumber	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-163: init_msg Information.

2.1.5.2.2 setup_msg

This routine completes the initialization of the message bar after realizing the widget.

Calls	
Function	**Where Described**
XtSetValues	public domain source of X, version 11, Release 4 (X11R4)
XtNumber	public domain source of X, version 11, Release 4 (X11R4)
XDefineCursor	public domain source of X, version 11, Release 4 (X11R4)
XtWindow	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-164: setup_msg Information.

2.1.5.2.3 put_msg

This routine puts a message on the message bar.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
format	pointer to char	Standard
arg1	int	Standard
arg2	int	Standard
arg3	int	Standard
arg4	int	Standard
arg5	int	Standard
Calls		
Function	**Where Described**	
msg_args		
XtSetValues	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-165: put_msg Information.

2.1.5.2.4 clear_message

This routine clears a message from the message bar.

Calls	
Function	**Where Described**
XtSetValues	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-166: clear_message Information.

2.1.5.2.5 blink_msg

This routine makes the message bar message blink.

Calls	
Function	**Where Described**
clear_message	this file

Table 2.1-167: blink_msg Information.

2.1.5.3 actions.c CSU Description (/simnet/cmd/nom/gui)

This CSU provides support for translating menu items into Action requests.

2.1.5.3.1 act_ActionReqPDUFill

This routine fills in the supplied PDU with the passed information. If no eventPDU is supplied, one is allocated and returned. The user must free the eventPDU later. The routine returns the MMP_PDUStruct if successful and 0 if unsuccessful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pdu	pointer to MMP_PDUStruct	
resourceID	long	Standard
actionID	long	Standard
Return Values		
Return Value	**Type**	**Meaning**
NULL	MMP_PDUStruct	unsuccessful
actionPDU	MMP_PDUStruct	successful; the filled in PDU structure

Table 2.1-168: act_ActionReqPDUFill Information.

2.1.5.3.2 act_SendActionToSelected

This routine generates an action request for each fo the selected pieces of equipment. *equip* is a pointer to the equipment in the nomEquipInfo table; *index* is the equipment index in the nomEquipInfo table.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
actionID	int	Standard
Calls		
Function	**Where Described**	
icon_IsSelected	icons.c	
act_SendAndCheckReq	this file	

Table 2.1-169: act_SendActionToSelected Information.

2.1.5.3.3 act_SendAndCheckReq

This routine is used to send an action request using a Transact. The possible responses are tested and, if necessary, reported to the user.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
dest	pointer to char	Standard
request	pointer to MMP_PDUStruct	
response	pointer to MMP_PDUStruct	
Calls		
Function	**Where Described**	
MMP_Transact	MMP.c	
DISPLAY_ERROR		

Table 2.1-170: act_SendAndCheckReq Information.

2.1.5.3.4 VCR_On

This routine is not used in the version 6.6 release.

2.1.5.3.5 VCR_Off

This routine is not used in the version 6.6 release.

2.1.5.3.6 VCR_Replay

This routine is not used in the version 6.6 release.

2.1.5.4 mmp.c CSU Description (/simnet/cmd/nom/gui)

This CSU includes functions that maintain information in the shared memory equipment table.

2.1.5.4.1 MMP_ProcessPDU

This routine is the central handler for all Manager-Manager PDUs received by the NOM graphical users interface. First the transfer syntax and the internal PDU syntax are translated. The routine performs further processing based on the kind of PDU received.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pdu	pointer to MMP_PDUStruct	
Calls		
Function	**Where Described**	
event_EventDispatch	events.c	

Table 2.1-171: MMP_ProcessPDU Information.

2.1.5.4.2 MMP_TracePDU

This routine is not used in the version 6.6 release.

2.1.5.4.3 MMP_SendPDU

This routine takes the passed internal representation of the MMP PDU, converts it into the transfer syntax and asynchronously sends it to the specified destination using the appropriate communication. This routine should only be used to send events or other PDUs not expecting replies. The routine returns 0 if successful and an error code if unsuccessful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
destination	pointer to char	Standard
pdu	pointer to MMP_PDUStruct	
Return Values		
Return Value	**Type**	**Meaning**
IPC_SendMessage(destination, mmProtocolKind, length, pdu)	ErrorCode	If 0, then successful; otherwise, unsuccessful
Calls		
Function	**Where Described**	
IPC_SendMessage	libipc in MCC CSCI	

Table 2.1-172: MMP_SendPDU Information.

2.1.5.4.4 MMP_SendRequest

This routine performs an asynchronous transaction not waiting for the response to an originating MMP PDU. The routine returns 0 if successful and an error code if unsuccessful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
destination	pointer to char	Standard
pdu	pointer to MMP_PDUStruct	
Return Values		
Return Value	**Type**	**Meaning**
IPC_SendRequest(destination, mmProtocolKind, length, pdu)	ErrorCode	If 0, then successful; otherwise, unsuccessful
Calls		
Function	**Where Described**	
IPC_SendRequest	libipc in MCC CSCI	

Table 2.1-173: MMP_SendRequest Information.

2.1.5.4.5 MMP_Transact

This routine performs a synchronous transaction waiting for the response to an originating MMP PDU. The routine returns 0 if successful and an error code if unsuccessful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
destination	pointer to char	Standard
pdu	pointer to MMP_PDUStruct	
response	pointer to MMP_PDUStruct	
Return Values		
Return Value	**Type**	**Meaning**
1	ErrorCode	unsuccessful
IPC_Transact(destintion, mmProtocolkind, length, pdu, rspKind, rspLen, response)	ErrorCode	If 0, then successful; otherwise, unsuccessful.

Table 2.1-174: MMP_Transact Information.

2.1.5.5 gui_ipc.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for gui.ipc.c. (See 2.1.5.1.)

2.1.5.6 msgsw.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for msgsw.c. (See 2.1.5.2.)

2.1.5.7 actions.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for actions.c (See 2.1.5.3.)

2.1.5.8 events.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for events.c. (See 2.1.3.1.)

2.1.5.9 equipment.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for equipment.c. (See 2.1.3.2.)

2.1.5.10 mmp.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for mmp.c. (See 2.1.5.4.)

2.1.6 Interface to Comms.CSC Description

This lower-level CSC formats and sends requests to Comms CSC to obtain a window on a remote SIM UNIX shell. This CSC uses local monitor protocol to communicate with Comms.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.1.6-1.

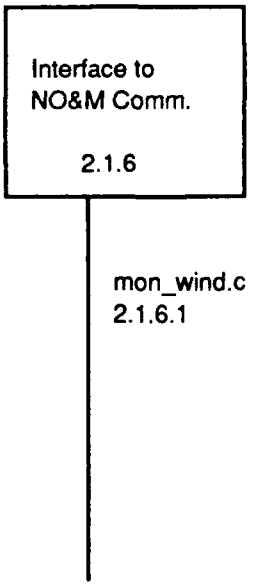


Figure 2.1.6-1: GUI--Interface to Comms.

2.1.6.1 mon_wind.c CSU Description (/simnet/cmd/monitor)

This CSU is used by GUI to obtain a "window" into the shell spawned by a remote rover process. While functionally relevant to GUI, this file is linked in NOM Comms process.

This CSU uses the simulation Ethernet address of the remote system, and the channel number of the remote shell.

Characters typed to mon_wind's stdin are passed immediately to the remote shell's stdin, without modification or local echoing (there are two exceptions noted below). Characters output by the remote shell to its stdout or stderr are conveyed to mon_wind and output to mon_wind's stdout, also without modification. Mon_wind recognizes two character sequences as special commands that it interprets rather than passing along to the remote shell.

When mon_wind is used to resume access to an existing, remote shell, the fido process local to mon_wind may provide mon_wind with a transcript of the shell's recent output. This session history is recorded by fido for all the remote shells it has been involved with, up to some maximum number of shells and some maximum number of characters per shell.

2.1.6.1.1 main

This routine is the program entry point. The command line arguments are parsed. The routine looks up the YUMM socket on which fido is listening. A free pseudo-tty for communicating with fido is opened and an attach message is sent to fido. The user's terminal is configured for raw I/O. The routine uses a small finite state machine to watch for escape sequences that terminate or suspend the tee_in process. The text is transferred between the user's terminal and fido and processed.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
argc	int	Standard
argv	pointer to array of char	Standard

Calls	
Function	**Where Described**
net_addr_str_to_bin	libnetif in MCC CSCI document
YUMMInit	libipc in MCC CSCI document
YUMMProcess	libipc in MCC CSCI
LkUpSocket	libipc in MCC CSCI
OpenPseudoTTY	common.c
SendMsg	libipc in MCC CSCI
SetTerminalRaw	common.c
FatalSystemError	common.c
RestoreTerminal	common.c

Table 2.1-175: main Information.

2.1.7 Shared Processing CSC Description

This lower-level CSC contains definitions and procedures used by more than one GUI CSC.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.1.7-1.

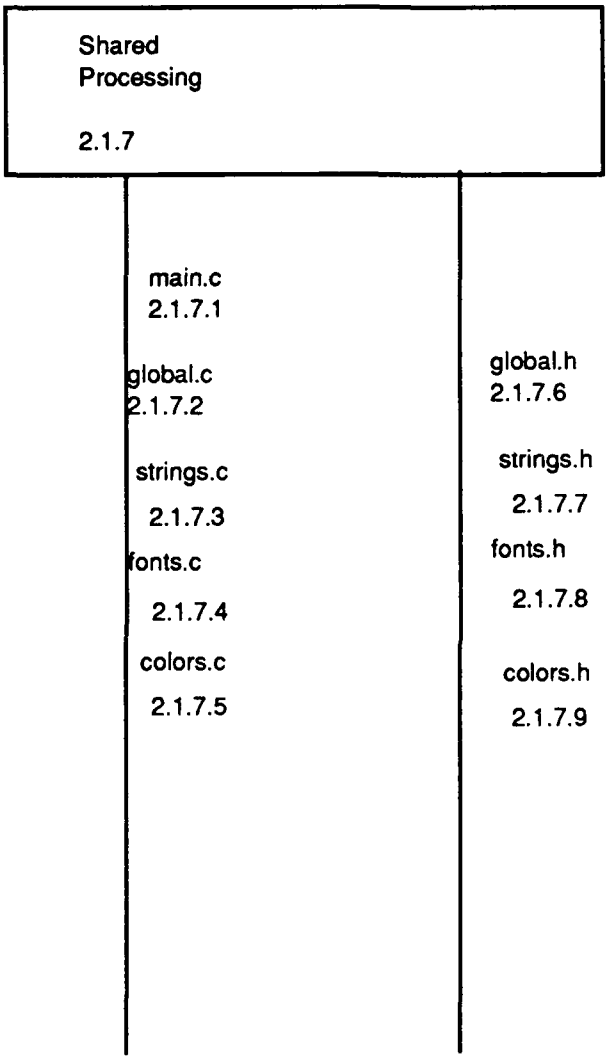


Figure 2.1.7-1 GUI--Shared Processing

2.1.7.1 main.c CSU Description (/simnet/cmd/nom/gui)

This CSU contains the entry p oint and main loop for GUI processing.

2.1.7.1 main

This routine contains the entry point to the GUI process. High priority for the process to prevent interruptions. All submodules of the GUI are initialized. The stored site depiction is loaded. Communications are opened with the server process and the main loop is entered. A handler is set up to process non-X events (asynchronous interrupts). The Xt main loop is invoked.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
argc	int	Standard
argv	pointer to array of char	Standard

Calls	
Function	**Where Described**
IPC_InitIPC	gui_ipc.c
XtInitialize	public domain source of X, version 11, Release 4 (X11R4)
XSetIoErrorHandler	public domain source of X, version 11, Release 4 (X11R4)
XtGetApplicationResources	public domain source of X, version 11, Release 4 (X11R4)
XtNumber	public domain source of X, version 11, Release 4 (X11R4)
XtDisplay	public domain source of X, version 11, Release 4 (X11R4)
XtScreen	public domain source of X, version 11, Release 4 (X11R4)
DefaultScreen	
DefaultGC	
tdb_init_cache	libtdb in MCC CSCI document
font_InitFonts	fonts.c
color_InitColors	colors.c
XtCreteManagedWidget	public domain source of X, version 11, Release 4 (X11R4)
panel_InitPanel	panels.c
canvas_InitCanvas	canvas.c
init_msg	msgsw.c
panel_SetupPanel	panel.c
canvas_SetupCanvas	canvas.c
XtSetArt	public domain source of X, version 11, Release 4 (X11R4)
XtSetValues	public domain source of X, version 11, Release 4 (X11R4)
file_LoadDefaultFile	file.c
IPC_SetupIPC	gui_ipc.c
XtAddTimeOut	public domain source of X, version 11, Release 4 (X11R4)
XtAddWorkProc	public domain source of X, version 11, Release 4 (X11R4)
XtMainLoop	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-176: main Information.

2.1.7.1.2 TimeoutInterrupts

This routine sets the timeout interrupts. The routine returns 0 if succesful.

Return Values		
Return Value	**Type**	**Meaning**
0	int	Successful

Calls	
Function	**Where Described**
XtAddTimeOut	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-177: TimeoutInterrupts Information.

2.1.7.1.3 WorkInterrupts

This routine sets the work interrupts.

Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	Unsuccessful

Table 2.1-178: WorkInterrupts Information.

2.1.7.1.4 MainLoop

This routine contains the main loop of the GUI process. MainLoop processes X events only if waiting, to prevent blocking. Asynchronous traps are allowed to get in during a window.

Calls	
Function	**Where Described**
XtAppPending	public domain source of X, version 11, Release 4 (X11R4)
XtAppNextEvent	public domain source of X, version 11, Release 4 (X11R4)
XtDefaultAppContext	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-179: MainLoop Information.

2.1.7.1.5 quit

This routine exits the GUI of the NOM and performs all necessary cleanup. The SIMNET libraries are cleaned up and terminated, the top level window is killed, and the process is exited.

Calls	
Function	**Where Described**
IPC_CleanUp	gui_ipc.c
tdb_terminate	libtdb in MCC CSCI document
XtDestroyWidget	public domain source of X, version 11, Release 4 (X11R4)

Table 2.1-180: quit Information.

2.1.7.2 global.c CSU Description (/simnet/cmd/nom/gui)

This CSU contains declarations of global variables use by GUI.

2.1.7.3 strings.c CSU Description (/simnet/cmd/nom/gui)

This CSU contains generic string utilities. The file contins relatively low-level string manipulation routines. While some perform useful processing, others are defined for clarity or to avoid common coding errors.

2.1.7.3.1 StringCreateCopy

This routine is a low level library routine to hide details while creating a new copy of an existing string.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
string	char	Standard
Return Values		
Return Value	**Type**	**Meaning**
new_copy	char	A new copy of string

Table 2.1-181: StringCreateCopy Information.

2.1.7.3.2 StringEqual

This routine uses the standard routine **strcmp** to determine if two strings are equal.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
str1	char	Standard
str2	char	Standard
Return Values		
Return Value	**Type**	**Meaning**
strcmp(str1, str2) == 0	int	If TRUE, the strings are equal; If FALSE, the strings are not equal.

Table 2.1-182: StringEqual Information.

2.1.7.3.3 StringCopyToken

This routine places a copy of the first word in the passed buffer and returns a pointer to the end of that word in the string.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
string	pointer to char	Standard
word	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
NULL	pointer to char	Unsuccessful
string + cur_char	pointer to char	pointer to the end of *word* in the string

Table 2.1-183: StringCopyToken Information.

2.1.7.3.4 StringContainsWord

This routine returns TRUE if the passed word is found as a separate word in the passed string, and FALSE if the word is not found. The routine assumes there are no words in the string of greater than size INFINITE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
string	pointer to char	*Standard*
word	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	*word* is a separate word in the string
FALSE	int	*word* is not found in the string.
Calls		
Function	**Where Described**	
StringCopyToken	this file	
StringEqual	string.c	

Table 2.1-184: StringContainsWord Information.

2.1.7.4 fonts.c CSU Description (/simnet/cmd/nom/gui)

This CSU creates and defines fonts used by GUI. The file creates a set of fonts which may then be referenced in X Arg lists for any widget.

2.1.7.4.1 font_InitFonts

This routine creates the pool of fonts for subsequent use as XtArgValues. The routine returns 0 if successful and an error code if unsuccessful.

Return Values		
Return Value	**Type**	**Meaning**
ERROR_NO_FONT	ErrorCode	Unsuccessful
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
XLoadQueryFont	public domain source of X, version 11, Release 4 (X11R4)	

Table 2.1-185: font_InitFonts Information.

2.1.7.5 colors.c CSU Description (/simnet/cmd/nom/gui)

This CSU creates and defines colors used by GUI.

2.1.7.5.1 color_InitColors

This routine creates the pool of colors for subsequent use as XtArg Values. The color cells are allocated from the server and the foreground and background colors are determined. The routine returns 0 if successful and an error code if unsuccessful.

Return Values		
Return Value	**Type**	**Meaning**
ERROR_NO_COLOR	ErrorCode	Unsuccessful
0	ErrorCode	Successful
Calls		
Function	**Where Described**	
XDefaultColorMap	public domain source of X, version 11, Release 4 (X11R4)	
XAllocNamedColor	public domain source of X, version 11, Release 4 (X11R4)	
WhitePixel		
BlackPixel		

Table 2.1-186: color_InitColors Information.

2.1.7.6 global.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for global.c. (See 2.1.7.2.)

2.1.7.7 strings.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for strings.c. (See 2.1.7.3.)

2.1.7.8 fonts.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for fonts.c. (See 2.1.7.4.)

2.1.7.9 colors.h CSU Description (/simnet/cmd/nom/gui)

This CSU is the interface file for colors.c. (See 2.1.7.5.)

2.2 MAP (Management Applications Process) CSC Description

A separate CSC, called the management application process (MAP), is responsible for actually performing the management tasks of the NOM.

This process normally monitors the state of running simulators and listens for any error reports which occur on simulators or MCCs, which it then logs on disk. MAP also monitors status PDUs for events and logs them into an Events Log on disk. Events which it determines are important are immediately communicated to GUI. All other information about simulators, along with certain NOM parameters, are maintained by the MAP only in shared memory for access by GUI.

MAP also accepts requests from the user interface to perform management tasks. Upon receiving a request, the MAP may coordinate the efforts of other NOM processes in addition to performing its own processing. The MAP communicates with the simulators via the SIMNET network for state information, interfaces to the power controller (using the separate lamplighter interface) through shared memory to determine the overall state of the simulator, and uses the communications subsystem to change simulator states.

Generally, the MAP will send an immediate acknowledgement to GUI when it receives a request and a subsequent event notifying the user interface if the request ultimately fails.

MAP has the major responsibility for maintaining shared memory. It initializes current threshold values, which can be updated by GUI, for equipment status values that generate alarms, and manages the simulator attributes table contained in shared memory.

Events:

MAP maintains a system-wide events log on disk, which a technician may display. When MAP determines an event to be 'important,' it sets an alarm bit in the appropriate shared memory field for the selected SIM; this causes a color-coded alarmed display. Any of the following events will set the alarm bit.

- Preset equipment threshold values are exceeded by current equipment conditions;
- SIM error report;
- power problem;
- A user request whose failure has been detected by MAP.

Equipment and power problems can be self-correcting; when corrected, MAP resets the alarm bit, causing the SIM alarmed state to clear. Problems associated with the other conditions above, are not self-correcting; a SIM alarmed display can only be cleared at the user interface. Alarmed events that self-correct remain in the outstanding events log until acknowledged by GUI.

A fully configured MAP implements the following management commands:

- Activate/Deactivate, which enters/removes a simulation from an exercise;
- Shutdown, which shuts down the SIM application and its UNIX operating system;
- Start/Stop, which starts/stops a simulation application process;
- Powerup/powerdown, which controls power to the simulator.

A functional breakdown of MAP CSC is illustrated in Figure 2.2-1.

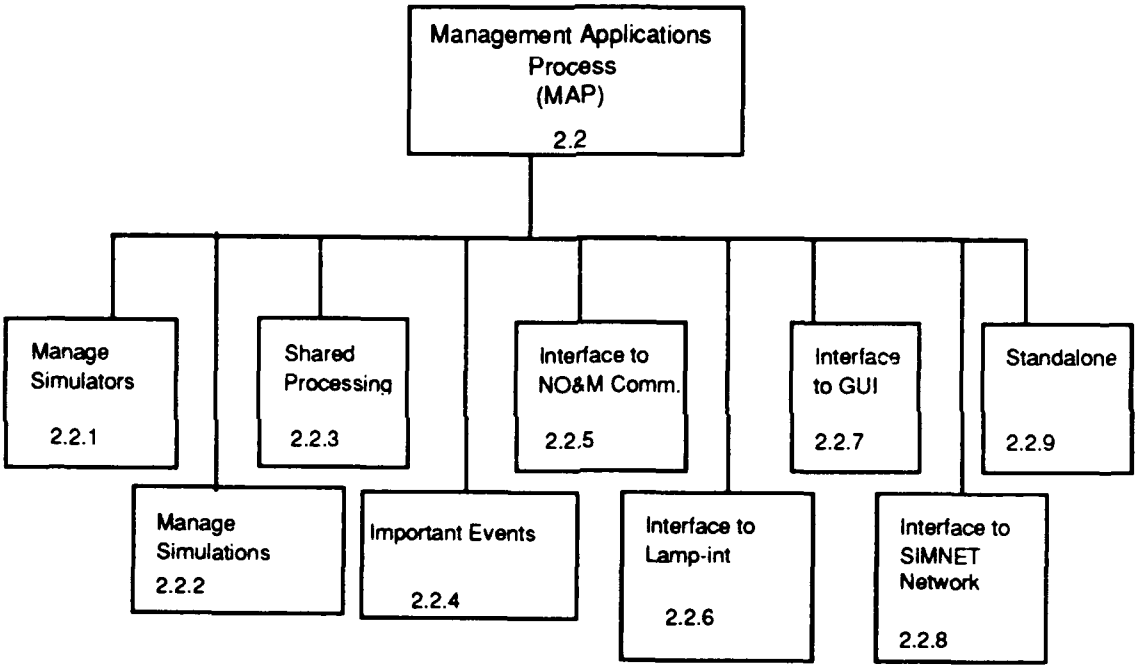


Figure 2.2-1: MAP CSC Structure.

2.2.1 Manage Simulators CSC Description

This lower-level CSC creates and sends GUI-requested commands on simulators. These commands are: powerup/powerdown, shutdown, and start/stop a simulator process.

This CSC also monitors simulators. From a shared memory table, it accesses power-related events PDUs placed there periodically by the Lamplighter Interface, and simulator equipment-related events placed there periodically by NOM-SIM communications. It also monitors SIM and MCC hosts for error reports. This CSC maintains error and event logs on disk, which are viewable by GUI.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.2.1-1.

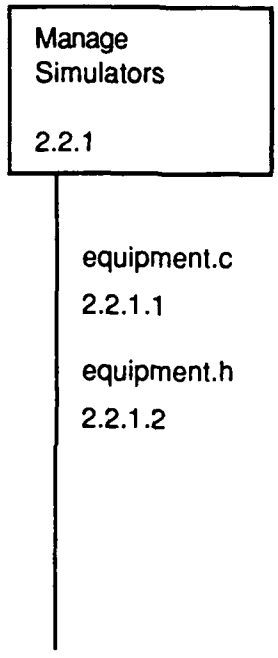


Figure 2.2.1-1: MAP--Manage Simulators.

2.2.1.1 equipment.c CSU Description (/simnet/cmd/nom/mini)

This CSU is devoted to performing host-related management. This includes tasks such as powering up or down the host, halting the operating system, and monitoring the equipment status (temperatures and voltages). This CSU includes functions that maintain information in the shared memory equipment table.

2.2.1.1.1 Equip_GetTrailerLoc

This routine returns the trailer location of the passed equipment from the Equipment Table.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
sEquiomentTable [index].info ->trailerLoc	pointer to char	The trailer location from the Equipment Table.

Table 2.2-1: Equip_GetTrailerLoc Information.

2.2.1.1.2 Equip_LogEvent

This routine formates the passed event text for this equipment and logs it. The routine does NOT add the event to the list of outstanding events. The formatted event and log time are provided as output parameters.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
eventText	pointer to char	Standard
formEvent	pointer to pointer to char	Standard
time	pointer to pointer to char	Standard
Calls		
Function	**Where Described**	
EM_LogEventText	events.c	

Table 2.2-2: Equip_LogEvent Information.

2.2.1.1.3 Equip_ProcessEvent

This routine processes an event received for this equipment. The event is an MM event, identified by a code. The event may have been generated internally by this process or received from another process or the CMC. The event is logged and forwarded.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
eventID	int	Standard

Calls	
Function	**Where Described**
UserReport	
Equip_AddEvent	equipment.c
EM_EventIsActFail	events.c
EM_SendNewStateEvent	events.c
EM_EventIsPower	

Table 2.2-3 Equip_ProcessEvent Information.

2.2.1.1.4 Event_AddEvent

This routine logs an event and adds it to the list of outstanding events. The event is logged and the formatted event and log time are obtained as output parameters. A new event record is created and initialized. The new event is added to the current event list.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
eventID	int	Standard
Calls		
Function	**Where Described**	
Equip_LogEvent	equipment.c	

Table 2.2-4: *Equip_AddEvent Information.*

2.2.1.1.15 Equip_ErrorReceived

This routine adds an event record from an ErrorReport PDU.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Calls		
Function	**Where Described**	
Equip_AddEvent	equipment.c	

Table 2.2-5: Equip_ErrorReceived Information.

2.2.1.1.16 Equip_ClearEvents

This routine clears any current alarm state in the list of current events.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Calls		
Function	**Where Described**	
EM_SendNewStateEvent	events.c	
Equip_LogEvent	equipment.c	

Table 2.2-6: Equip_ClearEvents Information.

2.2.1.1.17 Equip_EventStatus

This routine outputs the list of current events into the passed file.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
file	pointer to FILE	

Table 2.2-7 Equip_EventStatus Information.

2.2.1.1.18 Equip_Intialize

This routine is called to initialize INSTANCE variables. *strAddr* is a string version of the Ethernet address; *trailerLoc* is a string version of the floor location. The routine first determines the kind of equioment. The Equipment Table fields are initialized, and the UI is alerted of the new status.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
site	int	Standard
host	int	Standard
strAddr	pointer to char	Standard
trailerLoc	pointer to char	Standard
lampAddr	int	Standard
vehicleType	pointer to char	Standard
Calls		
Function	**Where Described**	
Sim_Initialize	simulators.c	
EM_SendNewStateEvent	events.c	
net_addr_str_to_bin	libnetif in MCC CSCI document	

Table 2.2-8: Equip_Initialize Information.

2.2.1.1.19 Equip_UpdateState

This routine is periodically called to update monitored state variables. Depending upon the type of PDUs received in the past interval and the internal NOM shared memory states, the operating state of the simulator is updated. A monitoring interval is the longest protocol interval used divided by EQUIP_PEEK_AHEAD.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Calls		
Function	**Where Described**	
EM_SendNewStateEvent	events.c	

Table 2.2-9: Equip_UpdateState Information.

2.2.1.1.20 Equip_EquipmentReceived

This routine is called when an equipment status PDU is received.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
pdu	pointer to ManagementPDU	
Calls		
Function	**Where Described**	
Equip_AddEvent	equipment.c	
EM_SendNewStateEvent	events.c	

Table 2.2-10: Equip_EquipmentReceived Information.

2.2.1.1.21 Equip_EquipmentStatus

This routine prints out the current equipment status.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
file	pointer to FILE	

Table 2.2-11: Equip_EquipmentStatus Information.

2.2.1.1.22 Equip_VehicleStatusReceived

This routine is called when a VehicleStatus PDU is received.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
pdu	pointer to DataCollectionPDU	
Calls		
Function	**Where Described**	
Sim_VehicleStatusReceived	simulators	

Table 2.2-12: Equip_VehicleStatusReceived Information.

2.2.1.1.23 Equip_ExerciseStatus

This routine prints out the current exercise status.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
file	pointer to FILE	
Calls		
Function	**Where Described**	
Sim_ExerciseStatus	simulators	

Table 2.2-13: Equip_ExerciseStatus Information.

2.2.1.1.24 Equip_Shutdown

This routine shuts down the host.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Calls		
Function	**Where Described**	
SendToRemoteHost	rsend.c	

Table 2.2-14: Equip_Shutdown Information.

2.2.1.1.25 EquipmentParseFile

This routine parses the MCC-pars file and creates instances for all the equipment and their simulators registered in the file.

Calls	
Function	**Where Described**
Equipment_initialize	this file

Table 2.2-15: EquipmentParseFile Information.

2.2.1.1.26 EquipmentInitTerrain

This routine initializes the terrain data base library (libtdb). The library is defaulted in the MCC-pars file, however it may be overridden through the command-line switch.

Calls	
Function	**Where Described**
tdb_init_cache	libtdb in MCC CSCI document
tdb_get_tdb_name	libtdb in MCC CSCI document

Table 2.2-16: EquipmentInitTerrain Information.

2.2.1.1.27 EquipmentInitialize

This routine initializes the equipment module. The lookup table for host indices is initialized, the MCC-pars file is read and the appropriate equipment records are recorded, and the terrain database is initialized.

Calls	
Function	**Where Described**
EquipmentParseFile	this file
EquipmentInitTerrain	this file

Table 2.2-17: EquipmentInitialize Information.

2.2.1.1.28 EquipmentDoMonitor

This routine monitors the receipt of Equipment Status reports and Vehicle Appearance Packets. Outstanding powerdowns are retried from this routine.

Calls	
Function	**Where Described**
Equip_UpdateState	this file
EquipmentRetryPowerdown	this file

Table 2.2-18: EquipmentDoMonitor Information.

2.2.1.1.29 TrailerLocToIndex

This routine is used to obtain the instance of a simulator matching the passed trailer and trailer element.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pair	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
index	int	The index of the simulator matching *pair*.
-1	int	Unsuccessful
Calls		
Function	**Where Described**	
Sim_Trailer	simulators.c	

Table 2.2-19: TrailerLocToIndex Information.

2.2.1.1.30 EquipmentRetryPowerdown

This routine attempts to powerdown any remaining simulatorts. The routine iterates through the sPowerdownEquiopment array searching for simulators waiting for powerdown. A non-zero entry impleies the corresponding simulator in the nomInfo Equipment Table should be powered down.

Calls	
Function	**Where Described**
Equip_ProcessEvent	this file

Table 2.2-20: EquipmentRetryPowerdown Information.

2.2.1.1.31 EquipmentPowerdown

This routine coordinates the multi-stage powerdown of multiple simulators.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
equipArray	pointer to char	Standard
Calls		
Function	**Where Described**	
Equip_Shutdown	this file	

Table 2.2-21: function Information.

2.2.1.2 equipment.h CSU Description (/simnet/cmd/nom/mini)

This CSU is the interface file for equipment.c.

2.2.2 Manage Simulations CSC Description

This lower-level CSC creates and sends GUI-requested commands on simulations. Such commands consists of activating/deactivating a simulation exercise. This CSC also monitors the state of a SIM application, and infers exercise status from vehicle state information provided by the Interface to the SIMNET Network (Section 2.2.8, below) and stores it in the shared memory table.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.2.2-1.

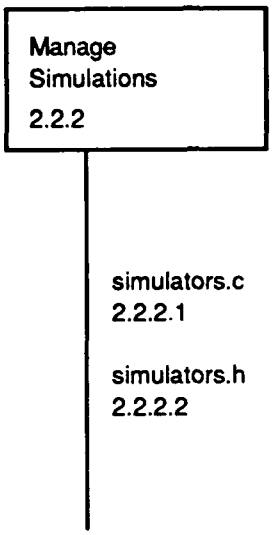


Figure 2.2.2-1: MAP--Manage Simulations

2.2.2.1 simulators.c CSU Description (/simnet/cmd/nom/mini)

This CSU is devoted to performing simulation software-related management. This module is the one which "speaks" the SIMNET simulation protocol, for monitoring simulation attributes, generating appropriate alarms, and issuing simulator software commands (to activate or deactivate, using Simulation protocol).

2.2.2.1.1 Sim_Initialize

This routine initializes the simulator variables. Some of the variables are set based on the passed parameters, while the user-initializable attributes are set to their default values.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
trailerLoc	pointer to char	Standard
info	pointer to NomEquipInfo	simulators.c

Table 2.2-22: Sim_Initialize Information.

2.2.2.1.2 Sim_TrailerMatch

This routine returns TRUE if the passed string represents the trailer ID of the vehicle.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
pair	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	*pair* represents the trailer ID
FALSE	int	*pair* does not represent the trailer ID

Table 2.2-23: Sim_TrailerMatch Information.

2.2.2.1.3 Sim_ExerciseStatus

This routine prints out the simulator's exercise status.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
file	pointer to FILE	

Table 2.2-24: Sim_ExerciseStatus Information.

2.2.2.1.4 Sim_VehicleStatusReceived

This routine is called when a Vehicle Status PDU is received in order to update the vehicle's attributes.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
pdu	pointer to DataCollectionPDU	

Table 2.2-25: Sim_VehicleStatusReceived Information.

2.2.2.1.5 Sim_SetActivateArgs

This routine interacts directly with the user to reset the default activate arguments. The user is prompted with the default for each of the arguments. The routine checks for validity of any input. The default will only be overwritten if the user inputs a value.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Calls		
Function	**Where Described**	
UserConfirm		
UserPrompt		
UserGetInput		
tdb_giv_xy_get_utm	libtdb in MCC CSCI	
tdb_giv_utm_get_xy	libtdb in MCC CSCI	
tdb_p_on_database	libtdb in MCC CSCI	
Sim_SetActivateArgs	simulators.c	

Table 2.2-26: Sim_SetActivateArgs Information.

2.2.2.1.6 Sim_GetActivateDefaults

This routine returns the default values used for activation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
rsp	pointer to MMP_PDUData	
Calls		
Function	**Where Described**	
Sim_PrepActivatePDU	simulators.c	

Table 2.2-27: Sim_GetActivateDefaults Information.

2.2.2.1.7 Sim_ActivateFromRequest

This routine attempts to activate this simulator using parameters passed in from an MM protocol request.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
mmPdu	pointer to MMP_PDUStruct	
Calls		
Function	**Where Described**	
AssocSendTransact	libassoc in MCC CSCI document	

Table 2.2-28: Sim_ActivateFromRequest Information.

2.2.2.1.8 Sim_PrepActivatePDU

This routine fills the planned activation values into the passed PDU for this simulator. This routine is used both for sending PDUs to the simulator and for returning default values to the GUI. The typical unit information, the generic status fields, the ground vehicle specific status field, the vehicle specific information, the constant fields, and the user settable fields are filled in.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
pdu	pointer to SimulationPDU	

Table 2.2-29: Sim_PrepActivatePDU Information.

2.2.2.1.9 Sim_Activate

This routine attempts to activate this simulator.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Calls		
Function	**Where Described**	
AssocSendTransact	libassoc in MCC CSCI document	
AssocError	libassoc in MCC CSCI document	

Table 2.2-30 Sim_Activate Information.

2.2.2.1.10 Sim_Deactivate

This routine attempts to deactivate this simulator.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Calls		
Function	**Where Described**	
AssocSendTransact	libassoc in MCC CSCI document	
AssocError	libassoc in MCC CSCI document	

Table 2.2-31 Sim_Deactivate Information.

2.2.2.1.11 Sim_Start

This routine attempts to run the simulator application.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Calls		
Function	**Where Described**	
SendToRemoteShell	rsend.c	

Table 2.2-32: Sim_Start Information.

2.2.2.1.12 Sim_Stop

This routine attempts to stop the simulator application.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Calls		
Function	**Where Described**	
SendToRemoteShell	rsend.c	

Table 2.2-33 Sim_Stop Information.

2.2.2.1.13 Sim_NoActivateRsp

This routine is a callback passed to the Association layer, and is called when there is no sign of an activate response from the requested simulator.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
data	pointer to char	Standard
length	long int	Standard
respondent	pointer to SimulationAddress	
param	pointer to long int	Standard
Calls		
Function	**Where Described**	
Equip_ProcessEvent	this file	

Table 2.2-34: Sim_NoActivateRsp Information.

2.2.2.1.14 Sim_NoDeactivateRsp

This routine is a callback passed to the Association layer, and is called when there is no sign of an deactivate response from the requested simulator.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
data	pointer to char	Standard
length	long int	Standard
respondent	pointer to SimulationAddress	
param	pointer to long int	Standard
Calls		
Function	**Where Described**	
Equip_ProcessEvent	this file	

Table 2.2-35: Sim_NoDeactivateRsp Information.

2.2.2.1.15 Sim_GotActivateRsp

This routine is a callback passed to the Association layer, and is called when an activate response is returned from the requested simulator.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
data	pointer to char	Standard
length	long int	Standard
respondent	pointer to SimulationAddress	
param	pointer to long int	Standard
Calls		
Function	**Where Described**	
Equip_ProcessEvent	this file	

Table 2.2-36: Sim_GotActivateRsp Information.

2.2.2.1.16 Sim_GotDeactivateRsp

This routine is a callback passed to the Association layer, and is called when a deactivate response is returned from the requested simulator.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
data	pointer to char	Standard
length	long int	Standard
respondent	pointer to SimulationAddress	
param	pointer to long int	Standard
Calls		
Function	**Where Described**	
Equip_ProcessEvent	this file	

Table 2.2-37: Sim_GotDeactivateRsp Information.

2.2.2.2 simulators.h CSU Description (/simnet/cmd/nom/mini)

This CSU is the interface file for simulators.c.

2.2.3 Shared Processing CSC Description

This lower-level CSC contains definitions and procedures used by more than one MAP CSC.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.2.3-1.

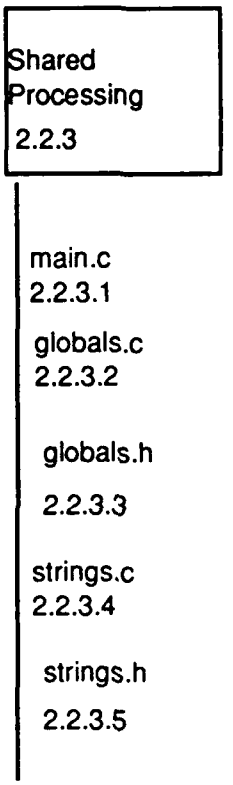


Figure 2.2.3-1: MAP--Shared Processing

2.2.3.1 main.c CSU Description (/simnet/cmd/nom/mini)

This CSU contains the entry point and main loop for MAP process.

2.2.3.1.1 main

This routine is the Management Application Process program entry point. The routine provides initial values of globals, which may be overridden by command line of parameters files. The command line is processed, performing any initial commands. The modules are initialized. Signals are set up, in order for termination to cause cleanup. Crash signals are set up for cleanup and core dump. The initial user input is prompted, and the master polling loop is started.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
argc	int	Standard
argv	pointer to pointer to char	Standard
Calls		
Function	**Where Described**	
IPC_InitIPC	gui_ipc.c	
GlobalsInitialize	globals.c	
MMP_Initialize		
ProcessCommandLine		
InitAlarms		
AlarmsEnabled	libipc.a in MCC CSCI	
DRN_InstallSpecialExceptions		
EM_InitEM	events.c	
EquipmentInitialize	equipment.c	
NetworkInitialize	pduio.c	
UIInitialize	tty_ui.c	
UI_MasterPoll	tty_ui.c	

Table 2.2-38: main Information.

2.2.3.2 globals.c CSU Description (/simnet/cmd/nom/mini)

This CSU contains global variables for inter-module communications.

2.2.3.2.1 GlobalsInitialize

This routine sets variable globals to their initial values, if necessary.

2.2.3.2.2 GlobalsCleanUp

This routine cleans up the globals, printing status information.

2.2.3.3 globals.h CSU Description (/simnet/cmd/nom/mini)

This CSU contains declarations for globals.c.

2.2.3.4 strings.c CSU Description (/simnet/cmd/nom/mini)

This CSU contains generic string utilities. The file contins relatively low-level string manipulation routines. While some perform useful processing, others are defined for clarity or to avoid common coding errors.

2.2.3.4.1 StringCreateCopy

This routine is a low level library routine to hide details while creating a new copy of an existing string.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
string	char	Standard
Return Values		
Return Value	**Type**	**Meaning**
new_copy	char	A new copy of string

Table 2.2-39 StringCreateCopy Information.

2.2.3.4.2 StringEqual

This routine uses the standard routine **strcmp** to determine if two strings are equal.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
str1	char	Standard
str2	char	Standard
Return Values		
Return Value	**Type**	**Meaning**
strcmp(str1, str2) == 0	int	If TRUE, the strings are equal; If FALSE, the strings are not equal.

Table 2.2-40: StringEqual Information.

2.2.3.4.3 StringCopyToken

This routine places a copy of the first word in the passed buffer and returns a pointer to the end of that word in the string.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
string	pointer to char	Standard
word	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
NULL	pointer to char	Unsuccessful
string + cur_char	pointer to char	pointer to the end of *word* in the string

Table 2.2-41: StringCopyToken Information.

2.2.3.4.4 StringContainsWord

This routine returns TRUE if the passed word is found as a separate word in the passed string, and FALSE if the word is not found. The routine assumes there are no words in the string of greater than size INFINITE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
string	pointer to char	Standard
word	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	*word* is a separate word in the string
FALSE	int	*word* is not found in the string.
Calls		
Function	**Where Described**	
StringCopyToken	strings.c	
StringEqual	strings.c	

Table 2.2-42: StringContainsWord Information.

2.2.3.5 strings.h CSU Description (/simnet/cmd/nom/mini)

This CSU displays help for current canvas operation (or error messages when appropriate).

2.2.4 Important Events CSC Description

This lower-level CSC determines which power- and equipment-related events should be treated as important events for notifying GUI. In making the determination which equipment-related events are outstanding, this CSC uses threshold values set by a technician. Also treated as an outstanding event is the lack of a SIM heartbeat, for which the NOM polls each SIM periodically. A monitor Response PDU from a SIM is the "heartbeat" that the NOM listens for in order to determine whether a SIM is running and connected to the network. The protocol for this communication function is the monitor protocol (refer to Section 2.2.5, "Interface to Comms.CSC Description.")

This CSC also formats error and outstanding event reports for sending to GUI.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.2.4-1. Additionally, this CSC references the CSUs, equipment.c and equip[illegible]nt.h, (Section 2.2.1.1 and Section 2.2.1.2).

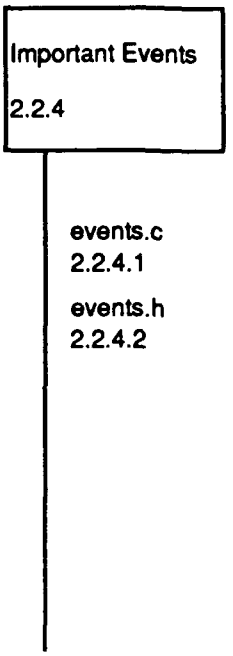


Figure 2.2.4-1: MAP--Important Events

2.2.4.1 events.c CSU Description (/simnet/cmd/nom/mini)

This CSU provides support for formatting and sending alarmed SIM state information to GUI. Notification of events is received, events are logged or consolidated, subscription requests are received, and events are forwarded to subscribers.

2.2.4.1.1 EM_InitEM

This routine initializes the EM module.

Calls	
Function	**Where Described**
EM_SubscriptionReceive	this file

Table 2.2-43 EM_InitEM Information.

2.2.4.1.2 EM_EventIsActFail

This routine returns whether the passed event indicates a request has failed.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
event	Event	events.h

Return Values		
Return Value	**Type**	**Meaning**
event_table [EM_iEventGetIndex(event)]. actFail	int	Whether the request has failed
Calls		
Function	**Where Described**	
EM_iEventGetIndex	this file	

Table 2.2-44: EM_EventIsActFail Information.

2.2.4.1.3 EM_EventIsActFail

This routine returns whether the passed event is related to power status in the lamplighter.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
event	Event	events.h
Return Values		
Return Value	**Type**	**Meaning**
event_table [EM_iEventGetIndex(event)]. power	int	Whether the event is related to power status.
Calls		
Function	**Where Described**	
EM_iEventGetIndex	this file	

Table 2.2-45: EM_EventIsPower Information.

2.2.4.1.4 EM_EventGetText

This routine returns the text used to display the event.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
event	Event	events.h
Return Values		
Return Value	**Type**	**Meaning**
event_table [EM_iEventGetIndex(event)]. eventText	int	The text used to display the event
Calls		
Function	**Where Described**	
EM_iEventGetIndex	this file	

Table 2.2-46: EM_EventGetText Information.

2.2.4.1.5 EM_EventReceive

This routine receives the event.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
eventPDU	pointer to MMP_PDUStruct	
Calls		
Function	**Where Described**	
MMP_SendPDU	mmp.c	

Table 2.2-47: EM_EventReceive Information.

2.2.4.1.6 EM_EventPDUFill

This routine fills the supplied PDU with the supplied information. If no eventPDU is supplied, one is allocated and returned, and the user must free it later. The routine returns a pointer to the filled in PDU.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pdu	pointer to MMP_PDUStruct	
eventID	long	Standard
resourceID	long	Standard
eventText	pointer to char	Standard

Return Values		
Return Value	**Type**	**Meaning**
NULL	pointer to MMP_PDUStruct	Unsuccessful
eventPDU	pointer to MMP_PDUStruct	the filled in PDU

Table 2.2-48 EM_EventPDUFill Information.

2.2.4.1.7 EM_SendNewStateEvent

This routine generates a state change event for the passed resource and forwards it to subscribers.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
resourceID	long	Standard
Return Values		
Return Value	**Type**	**Meaning**
0	int	Successful
1	int	Unsuccessful

Calls	
Function	**Where Described**
EM_EventPDUFill	this file
EM_EventReceive	this file

Table 2.2-49: EM_SendNewStateEvent Information.

2.2.4.1.9 EM_SubscriptionReceive

This routine accepts a subscription request and adds it to the list of active subscriptions.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
subscription	pointer to Subscription	
Calls		
Function	**Where Described**	
EM_iCopySubscription	this file	

Table 2.2-50: EM_SubscriptionReceive Information.

2.2.4.1.10 EM_LogEventText

This routine logs the passed event text into the event log. If the time is not NULL, the time is written into the event log in an ASCII representation of the log time.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
text	pointer to char	Standard
time	pointer to char	Standard
Calls		
Function	**Where Described**	
EM_iGetTimeField	this file	

Table 2.2-51: EM_LogEventText Information.

2.2.4.1.11 EM_CleanUp

This routine performs any necessary cleanup for the EM module.

2.2.4.1.12 EM_iGetTimeField

This internal routine returns a static string containing the current time formatted correctly for the event log.

Return Values		
Return Value	**Type**	**Meaning**
0	pointer to char	Unsuccessful
timeBuf	pointer to char	The formatted current time

Table 2.2-52 EM_iGetTimeField Information.

2.2.4.1.13 EM_iCopySubscriptions

This routine returns a newly memory allocated copy of the passed event Subscription.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
old	pointer to Subscription	
Return Values		
Return Value	**Type**	**Meaning**
new	pointer to Subscription	The memory allocated copy of the event Subscription.
Calls		
Function	**Where Described**	
StringCreateCopy	strings.c	

Table 2.2-53: function Information.

2.2.4.1.14 EM_iEventGetIndex

This routine returns the proper index into the event table for the passed event.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
event	Event	
Return Values		
Return Value	**Type**	**Meaning**
index	short	The index into the event table for *event*.

Table 2.2-54: EM_iEventGetIndex Information.

2.2.4.2 events.h CSU Description (/simnet/cmd/nom/mini)

This CSU is the interface file for events.c.

2.2.5 Interface to Comms. CSC Description

This lower-level CSC provides an interface between MAP and simulators. This interface uses two protocols, shared memory, and a local monitor protocol. This interface accesses SIM status information in shared memory which enables MAP to determine whether a SIM host is running and connected to the SIMNET network. The monitor protocol defines the message PDUs used in polling for a SIM "heartbeat." It also defines the message formats that allow MAP to pipe management commands to remote simulators.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.2.5-1. Additionally, this CSC cross-references the equipment.c CSU (section 2.2.1.1), monipc.h CSU(Section 2.4.4.5), and mon_rmcd.c (Section 2.4.5.1).

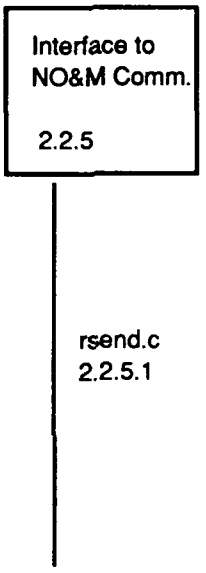


Figure 2.2.5-1: MAP--Interface to Comms.

2.2.5.1 rsend.c CSU Description (/simnet/cmd/nom/mini)

This CSU is a high-level interface for sending text strings to remote simulators via the Comms. CSC. It uses the protocol defined in monipc.h to make requests of the Comms. CSC. Specifically, this sends Unix commands to Masscomps such as sync and reboot.

2.2.5.1.1 SendToRemoteShell

This routine sends a text string to a shell executing on a remote simulator.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
address	pointer to NetworkAddress	
text	pointer to char	Standard
channel	int	Standard
Calls		
Function	**Where Described**	
LkUpSocket	libipc in MCC CSCI	
SendMsg	libipc in MCC CSCI	

Table 2.2-55: SendToRemoteShell Information.

2.2.5.1.2 SendToRemoteHost

This routine sends a command to the remote host, regardless of the channel.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
address	pointer to NetworkAddress	
text	pointer to char	Standard
Calls		
Function	**Where Described**	
LkUpSocket	libipc in MCC CSCI	
SendMsg	libipc in MCC CSCI	

Table 2.2-56: SendToRemoteHost Information.

2.2.6 Interface to Lamplighter-int CSC Description

This lower-level CSC receives power events from the Lamplighter Interface by accessing shared memory, and sends power requests to the Lamplighter Interface using Manager-Manager protocol.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.2.6-1. Additionally, this CSC references CSU mini_ipc.c (Section 2.2.7.3).

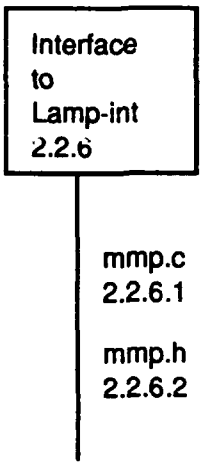


Figure 2.2.6-1 MAP--Interface to Lamplighter-int.

2.2.6.1 mmp.c CSU Description (/simnet/cmd/nom/mini)

This CSU contains support for sending and receiving Manager-Manager Protocol.

2.2.6.1.1 MMP_ProcessPDU

This routine is the central handler for all Manager-Manager PDUs received by the NOM Management Application server process. First the transfer syntax and the internal PDU syntax are translated. The routine performs further processing based on the kind of PDU received. *requestID* identifies the IPC transaction.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
psu	pointer to MMP_PDUStruct	
requestID	int	Standard
Calls		
Function	**Where Described**	
MMP_ProcessEvent	MMP.c	
MMP_DispatchActionReq	MMP.c	

Table 2.2-57: MMP_ProcessPDU Information.

2.2.6.1.2 MMP_DispatchActionReq

This routine dispatches all management action requests. *requestID* identifies the IPC transaction.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pdu	pointer to MMP_PDUStruct	
requestID	int	Standard
Calls		
Function	**Where Described**	
EquipmentPowerdown	equipment.c	
MMP_SendPDU	MMP.c	
Sim_ActivateFromRequest	simulators.c	
MMP_ActionToFunction		
Sim_GetActivateDefaults	simulators.c	

Table 2.2-58: MMP_DispatchActionReq Information.

2.2.6.1.3 MMP_ProcessEvent

This routine processes all events received via Management-Management protocol.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pdu	pointer to MMP_PDUStruct	
Calls		
Function	**Where Described**	
Equip_ProcessEvent	equipment.c	

Table 2.2-59: MMP_ProcessEvent Information.

2.2.6.1.4 MMP_SendPDU

This routine takes the passed internal representation of the MMP PDU, converts it into the transfer syntax and asynchronously sends it to the specified destination using the appropriate communication.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
destination	pointer to char	Standard
pdu	pointer to MMP_PDUStruct	
Calls		
Function	**Where Described**	
IPC_SendMessage	libipc in MCC CSCI	

Table 2.2-60 MMP_SendPDU Information.

2.2.6.2 mmp.h CSU Description (/simnet/cmd/nom/mini)

This is the interface file for mmp.c.

2.2.7 Interface to GUI CSC Description

This lower-level CSC accepts GUI requests which it formats for dispatching to the Lamplighter Interface and Comms CSCs. It also sends error reports to GUI. This CSC sends an acknowledgement to GUI upon receiving a request, and returns a failure message if the requested action failed.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.2.7-1. Additionally, this CSC references CSUs mmp.c and mmp.h (Section 2.2.6.1).

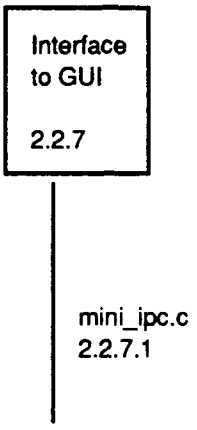


Figure 2.2.7-1: MAP--Interface to GUI.

2.2.7.1 mini_ipc.c CSU Description (/simnet/cmd/nom/mini)

This CSU provides higher-level interface for IPC.

2.2.7.1.1 IPC_InitIPC

This routine initializes the IPC for the NOM Graphical User Inteface. Sockets and handlers are established for receiving messages. The NOM shared memory is attached and initialized. This routine opens communications with the server process. The YUMM usage is initialized. The NOM is restricted to one GUI running at a time. If a socket has this process name associated with it, it must be deleted (the process probably died without closing). The message handler is registered to receive messages.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
procName	pointer to char	Standard
Calls		
Function	**Where Described**	
AttachSharedMem		
YUMMInit	libipc in MCC CSCI	
YUMMProcess	libipc in MCC CSCI	
LkUpSocket	libipc in MCC CSCI	
CloseSocket	libipc in MCC CSCI	
OpenSocket	libipc in MCC CSCI	

Table 2.2-61: IPC_InitIPC Information.

2.2.7.1.2 IPC_ProcessMessage

This routine is the YUMM message receipt handler for any protocol type. Alarms are disabled while messages are being processed, then reenabled when finished. The message is dispatched to the proper protocol handler.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	int	Standard
kind	long	Standard
length	int	Standard
data	pointer to char	Standard
requestID	long	Standard
Calls		
Function	**Where Described**	
MMP_ProcessPDU	MMP.c	
AlarmsEnabled	libipc in MCC CSCI	

Table 2.2-62 IPC_ProcessMessage Information.

2.2.7.1.3 IPC_TraceMessage

This routine is the handler used when tracing YUMM messages. The routine assumes the message is a NOM MMP PDU.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	char	Standard
tid	char	Standard
dstSkt	YUMMSocket	libipc
rspSkt	YUMMSocket	libipc
kind	long	Standard
length	int	Standard
data	pointer to char	Standard
Calls		
Function	**Where Described**	
MMP_TracePDU	MMP.c	

Table 2.2-63: IPC_TraceMessage Information.

2.2.7.1.4 IPC_SendMessage

This routine sends data to the specified process using YUMM IPC. Each time a message is sent, a YUMM lookup is performed in order to find any changes in the sockets. The routine returns 0 if successful and an error code if unsuccessful.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
destination	pointer to char	Stnadard
kind	long	Standard
length	int	Standard
data	pointer to char	Standard
Calls		
Function	**Where Described**	
LkUpSocket	libipc in MCC CSCI	
SendMsg	libipc in MCC CSCI	

Table 2.2-64: IPC_SendMessage Information.

2.2.8 Interface to SIMNET Network CSC Description

This CSC currently uses Simulation, Data Collection, and Management SIMNET protocols to communicate with simulator hosts. These protocols are described in the BBN Report listed earlier, titled "SIMNET Network and Protocols." Vehicle PDUs are received by this CSC, filtered, and stored in shared memory.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.2.8-1.

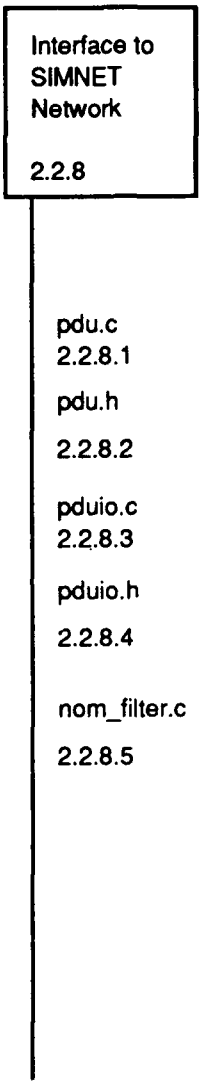


Figure 2.2.8-1: MAP Interface to Simnet Network.

2.2.8.1 pdu.c CSU Description (/simnet/cmd/nom/mini)

This CSU provides support for sending and receiving SIMNET PDUs. This file contains routines for processing SIMNET PDUs.

2.2.8.1.1 PDU_ProcessPDU

This is the central PDU processing dispatch routine. It handles all PDUs received from the CMC net. This uses supporting subroutines in this module to dispatch PDUs from the different protocols (e.g, sim, mgmgt, data).

2.2.8.1 PDU_ProcessPDU

Parameters		
Parameter	**Type**	**Where Typedef Declared**
assoc	pointer to AssociationPDU	
Calls		
Function	**Where Described**	
PDU_ProcessMgmtPDU	this file	
PDU_ProcessDataPDU	this file	
DRN_PrintAssociationPDU		

Table 2.2-65: PDU_ProcessPDU Information.

2.2.8.1.2 PDU_ProcessDataPDU

This function processes a datagram PDU pertaining to the Simulation Protocol.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
assoc	pointer to AssociationPDU	
site	int	Standard
host	int	Standard
Calls		
Function	**Where Described**	
Equip_VehicleStatusReceived	equipment.c	

Table 2.2-66: PDU_ProcessDataPDU Information.

2.2.8.1.3 PDU_ProcessMgmtPDU

This function processes a PDU pertaining to the Management Protocol. It gets a handle on the equipment index for this PDU. This function processes all error report PDUs; if there is no equip index, the PDU is ignored-- it is either from a remote site or a tape being played back. This function dispatches, using the appropriate message to the equipment index.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
assoc	pointer to AssociationPDU	

Calls	
Function	**Where Described**
EM_LogEventText	events.c
Equip_EquipmentReceived	equipment.c
Equip_ErrorReceived	equipment.c

Table 2.2-67: PDU_ProcessMgmtPDU Information.

2.2.8.2 pdu.h CSU Description (/simnet/cmd/nom/mini)

This is the interface file for pdu.c.

2.2.8.3 pduio.c CSU Description (/simnet/cmd/nom/mini)

This CSU initializes, polls, and shuts down the network interface. This CSU provides a service interface for MAP to send a request, and processes incoming SIMNET PDUs. This file front-ends the network interface at the PDU level.

2.2.8.3.1 NetworkInitialize

This function initializes the interface to the network for reading and writing PDUs. This includes initializing variables and the association layer of the SIMNET network.

Calls	
Function	**Where Described**
AssocError	libassoc in MCC CSCI document
net_stop	libnetif in MCC CSCI document
net_prom	libnetif in MCC CSCI document
net_flush	libnetif in MCC CSCI document
AssocGetSimAddress	libassoc in MCC CSCI document
AssocOpen	libassoc in MCC CSCI document

Table 2.2-68: NetworkInitialize Information.

2.2.8.3.2 PollNetwork

This function polls the network for any queued PDUs waiting to be heard. It returns whether there is a backlog of PDUs. This function declares the global variable, int errno. This function periodically sends a network handle to the association layer, sets a high priority to avoid being interrupted, and processes n PDUs at a time, to allow other periodic processing.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	there is a backlog of PDUs
FALSE	int	there is no backlog of PDUs

Calls	
Function	**Where Described**
AssocTickAssocLayer	libassoc.a in MCC CSCI
AssocReceiveAssocPDU	libassoc.a in MCC CSCI
PDU_ProcessPDU	pdu.c

Table 2.2-69: PollNetwork Information.

2.2.8.3.3 NetworkClose

This leaves the network interface in the approved condition.

Calls	
Function	**Where Described**
net_close	libnetif in MCC CSCI document

Table 2.2-70: NetworkClose Information.

2.2.8.4 pduio.h CSU Description (/simnet/cmd/nom/mini)

This is the interface file for pduio.c.

2.2.8.5 nom_filter.c CSU Description (/simnet/cmd/nom/mini)

This file contains NOM-specific filter code; it is used to get adequate performance out of the NOM. Currently this filters out Appearance PDUs in addition to NOM-irrelevant PDUs. This function is downloaded onto CMC card during initialization.

2.2.8.5.1 do_init

This code is predefined for reference by special code on the CMC card.

2.2.8.5.2 do_ioctl

This code is predefined for reference by special code on the CMC card.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
p	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
0	int	Successful

Table 2.2-71: do_ioctl Information.

2.2.8.5.3 do_tick

This code is predefined for reference by special code on the CMC card.

2.2.8.5.4 do_packet_from_host

This function sends a packet out on the network.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to short	Standard
Return Values		
Return Value	**Type**	**Meaning**
SEND_PACKET	int	Whether the packet was successfully sent

Table 2.2-72: do_packet_from_host Information.

2.2.8.5.5 do_packet_from_network

This function reads and processes both 2.0 and 802.3 packets.from simulators. The data are not aggregating PDUs, so it can be assumed that a PDU is a single association PDU. This function examines the association layer header to see where the data is stored, depending upon whether it is in a datagram, request, or response. It selects the kinds of PDUs the NOM is interested in--they are: Data Collection, Vehicle Status, management, and activate/deactivateResponse (protocols are described in "SIMNET Network and Protocols,"-- listed under "Related Documents," Section 1.7.).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to NetworkPacket	
Return Values		
Return Value	**Type**	**Meaning**
SEND_PACKET	int	The PDU is one of the PDU types the NOM is interested in
FREE_PACKET	int	The PDU only contains padding, or the NOM is not interested in the packet
Calls		
Function	**Where Described**	
GET_DATA_PTR		

Table 2.2-73: do_packet_from_network Information.

2.2.9 Standalone CSC Description

This CSC enables MAP to be used as a standalone utility, accepting commands directly from the terminal without GUI support.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.2.9-1

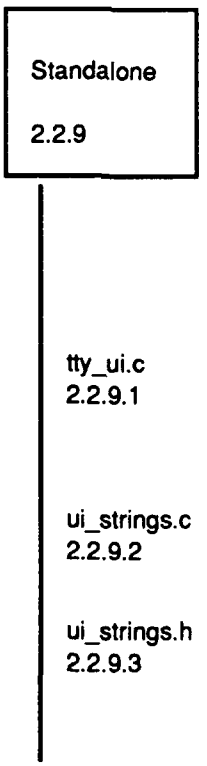


Figure 2.2.9-1: MAP--Standalone

2.2.9.1 tty_ui.c CSU Description (/simnet/cmd/nom/mini)

This CSU actively interfaces to the terminal to garner input commands and parse them. It then interfaces in a similar manner as mmp.c to the functionality provided by the MAP. This file provides ASCII terminal interface to MAP. The typename 'command' is typedefined in this file.

2.2.9.1.1 UIInitialize

This function initializes UI. It sets flags to configure the UI for non-blocking reads which do not wait for input.

2.2.9.1.2 UIRestore

This function restores UI to initial settings.

2.2.9.1.3 PollBackgroundTasks

This function checks for alarms, PDUs, and internal maintenance. This function allows some alarms to slip in by calling astpause. This call will also allow other processes to slip in if MAP is idle.

Calls	
Function	**Where Described**
AlarmsEnabled	libipc in MCC CSCI
PollNetwork	pduio.c
tickCount	
EquipmentDoMonitor	equipment.c

Table 2.2-74: PollBackgroundTasks Information.

2.2.9.1.4 UI_MasterPoll

This function is used in synchronizing all periodic tasks. It checks for input from the user.

Calls	
Function	**Where Described**
UserPrompt	
PollBackgroundTasks	this file
PollUser	this file
RunCommand	

Table 2.2-75: UI_MasterPoll Information.

2.2.9.1.5 PollUser

This function does a non-blocking read for user input. The routine returns FALSE if no input is ready. Currently, as there is never user input, this function will never return TRUE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
buffer	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	no input is ready
FALSE	int	input is ready
Calls		
Function	**Where Described**	
GetLine		

Table 2.2-76: PollUser Information.

2.2.9.2 ui_strings.c CSU Description (/simnet/cmd/nom/mini)

This CSU provides strings which are displayed to the user to represent different parameters and values.

This file contains strings used in User Interface. It includes strings for command keywords, help text. This file also includes text for reporting events, printing out equipment status, detailing simulator state, and confirm operations, and maintenance file output by the NOM.

2.2.9.3 ui_strings.h CSU Description (/simnet/cmd/nom/mini)

This is the interface file for ui_strings.c.

2.3 Lamplighter Interface CSC Description

The Lamplighter CSC interfaces to the power controller. This process also normally acts as a deamon, polling the power status of simulators regularly, and reporting to the MAP whenever an anomaly is detected. Lamp-int translates requests which it receives in messages into the format accepted by the power controller. Lamplighter power reports are evaluated by MAP, using operator-set threshold values, to determine whether an outstanding event has occurred.

A functional breakdown of Lamplighter Interface is illustrated in Figure 2.3-1.

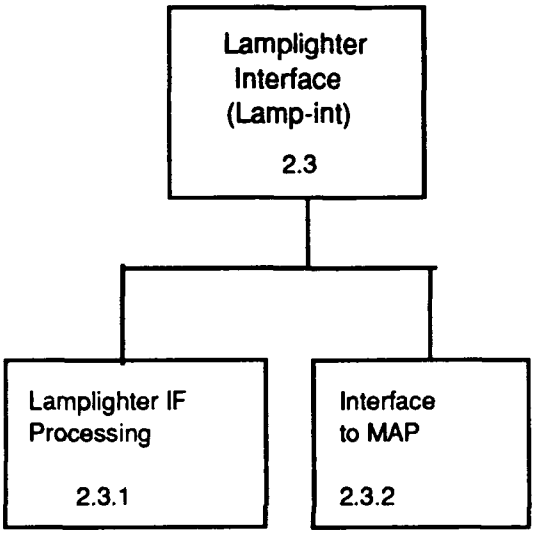


Figure 2.3-1: Lamplighter Interface CSC Structure.

2.3.1 Lamplighter Interface Processing CSC Description

This lower-level CSC polls the power controller for power status. MAP powerup/powerdown command requests which are sent to the higher-level CSC are translated by this lower-level CSC into serial messages which also dispatches the messages to the power controller.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.3.1-1.

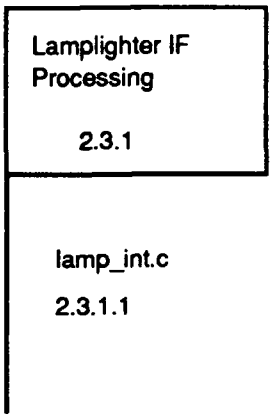


Figure 2.3.1-1: Lamplighter Interface--Lamplighter Processing.

2.3.1.1 lamp_int.c CSU Description (/simnet/cmd/nom/lamp)

This file contains functions that interface to a lamplighter control unit and to NOM shared memory.

2.3.1.1.1 main

This is the main function for lamp process. It uses IPC protocol to set up communications with MAP, using IPC protocol.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
argc	int	Standard
argv	pointer to array of char	Standard
Calls		
Function	**Where Described**	
YUMMInit	libipc in MCC CSCI	
YUMMProcess	libipc in MCC CSCI	
LkUpSocket	libipc in MCC CSCI	
CloseSocket	libipc in MCC CSCI	
OpenSocket	libipc in MCC CSCI	
add_lampName	shmem_acc.c	
main_loop	this file	
YUMMCleanUp	libipc in MCC CSCI	
InitAlarms		
init_lampnet	this file	

Table 2.3-1: main Information.

2.3.1.1.2 main_loop

This function is the main loop of this file. It checks the power state of simulators.

Calls	
Function	**Where Described**
poll_one_lamp	this file
send_on	this file
send_off	this file
poll_lampnet	this file

Table 2.3-2: main_loop Information.

2.3.1.1.3 init_lampnet

This function initializes an interface with a lamplighter device RS232 port. Lamp_dev is configured to ignore break conditions, to ignore parity errors (as the protocol between the NOM and the lamp_int is not ASCII), to run at 1200 baud, to use 8-bit characters, to enable the receiver, to use a local (non-modem) line, to read at least three characters, and to wait up to .1 seconds on a read before returning. For detailed configuration information, see the UNIX man page "termio".

Calls	
Function	**Where Described**
get_lamp_dev	shmem_acc.c
attach_nom_table	shmem_acc.c

Table 2.3-3: init_lampnet Information.

2.3.1.1.4 process_a_msg

This function reads and processes a message from a lamplighter device. This message includes SIM equipment state information such as: smoke, emergency, ready state, and on-state. Message contents are stored in shared memory.

Calls	
Function	**Where Described**
find_shmem_index	shmem_acc.c
get_smokestate	shmem_acc.c
send_mini_ipc	this file
get_readystate	shmem_acc.c
get_emstate	shmem_acc.c
set_powerstate	shmem_acc.c
set_readystate	shmem_acc.c
set_smokestate	shmem_acc.c
set_emstate	shmem_acc.c

Table 2.3-4: process_a_msg Information.

2.3.1.1.5 ipc_process

This function processes request for status of requested actions..

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	long	Standard
msg_kind	long	Standard
msg_len	long	Standard
dummy	long	Standard
msg_data	pointer to char	Standard
Calls		
Function	**Where Described**	
AlarmsEnabled	libipc.a in MCC CSCI	
send_on	this file	
get_lamp_addr	shmem_acc.c	

Table 2.3-5: ipc_process Information.

2.3.1.1.6 poll_lampnet

This function polls the lampdevice for SIM equipment input.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
p	long	Standard
a	Alarm	
Calls		
Function	**Where Described**	
set_powerstate	shmem_acc.c	
put_msg	msgsw.c	
get_lamp_addr	shmem_acc.c	
process_a_msg	this file	
SetAlarm		

Table 2.3-6: poll_lampnet Information.

2.3.1.1.7 poll_one_lamp

This function polls for input from one SIM.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
num	int	Standard
Calls		
Function	**Where Described**	
process_a_msg	this file	

Table 2.3-7: poll_one_lamp Information.

2.3.1.1.8 put_msg

This function puts out a message to the lamplighter device.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
addr	char	Standard
cmd	char	Standard
Calls		
Function	**Where Described**	
send_off	this file	

Table 2.3-8: put_msg Information.

2.3.1.1.9 send_on

This function puts out a SIM on state message to the lamplighter device.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
addr	int	Standard
Calls		
Function	**Where Described**	
put_msg	msgsw.c	

Table 2.3-9: send_on Information.

2.3.1.1.10 send_off

This function puts out a SIM off message to the lamplighter device.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
addr	int	Standard
Calls		
Function	**Where Described**	
put_msg	msgsw.c	

Table 2.3-10: send_off Information.

2.3.1.1.11 send_e_off

This function puts out an emergency off state message to the lamplighter device.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
addr	int	Standard
Calls		
Function	**Where Described**	
put_msg	msgsw.c	

Table 2.3-11: send_e_off Information.

2.3.1.1.12 add_ether

This function is not used in the version 6.6 release.

2.3.1.1.13 send_mini_ipc

This function sends an IPC message to MAP about an event.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
eveid	int	Standard
node	int	Standard
Calls		
Function	**Where Described**	
LkUpSocket	libipc in MCC CSCI	
SendMsg	libipc in MCC CSCI	

Table 2.3-12: send_mini_ipc Information.

2.3.1.1.14 itoa

This function returns a pointer to a string returned by a function similar to the "C" function, itoa.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
n	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
s	pointer to char	String value

Table 2.3-13: itoa Information.

2.3.2 Interface to MAP CSC Description

This lower-level CSC formats and stores power events into the shared memory table.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.3.2-1.

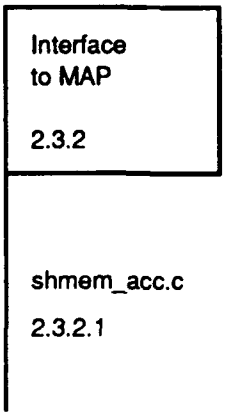


Figure 2.3.2-1: Lamplighter Interface--Interface to MAP.

2.3.2.1 shmem_acc.c CSU Description (/simnet/cmd/nom/lamp)

This CSU is also used by the CSCs in MAP to access shared memory. This file contains functions that provide access to various fields in the struct, nomEquipTable, which is the NOM shared memory table.

2.3.2.1.1 attach_nom_table

This function accesses nomEquipTable in shared memory.

Calls	
Function	**Where Described**
AttachSharedMem	libipc in MCC CSCI

Table 2.3-14: attach_nom_table Information.

2.3.2.1.2 add_lampName

This function copies the name of a lamplighter procedure into 'lampProcName'.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
s	pointer to char	Standard

Table 2.3-15: add_lampName Information.

2.3.2.1.3 get_lamp_dev

This function returns a pointer to the character value returned by get_lamp_dev, which is the contents of tty_dev.

Return Values		
Return Value	**Type**	**Meaning**
nonInfo->tty_dev	pointer to char	contents of tty_dev

Table 2.3-16: get_lamp_dev Information.

2.3.2.1.4 get_numEquip

This function returns the contents of numEquip.

Return Values		
Return Value	**Type**	**Meaning**
nonInfo->numEquip	int	contents of numEquip

Table 2.3-17: get_lamp_dev Information.

2.3.2.1.5 get_lamp_addr

This function returns the contents of lamp_addr indexed by i.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
i	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
nomInfo -> equipTable[i].lamp_addr)	int	the contents of lamp_addr

Table 2.3-18: get_lamp_addr Information.

2.3.2.1.6 get_lamp_timeout()

This function returns the contents of timeout.

Return Values		
Return Value	**Type**	**Meaning**
nomInfo->timeout	int	the contents of timeout

2.3.2.1.7 set_powerstate

This function writes value of s into the location of powerState indexed by j.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
j	int	Standard
s	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
nomInfo-> equipTable[j].powerState = s	int	the contents of the powerState indexed by j

Table 2.3-19: set_powerstate Information.

2.3.2.1.8 get_powerstate

This function returns the contents of powerState indexed by j.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
j	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
nomInfo -> equipTable[j].powerState	int	the contents of powerState indexed by j

Table 2.3-20: get_powerstate Information.

2.3.2.1.9 set_readystate

This function writes value of s into the location of readystate indexed by j.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
s	int	Standard
j	int	Standard

Table 2.3-21: set_readystate Information.

2.3.2.1.10 get_readystate

This function returns the contents of readystate indexed by j.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
j	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
nomInfo -> equipTable[j].readyState	int	the contents of readyState indexed by j

Table 2.3-22: get_readystate Information.

2.3.2.1.11 set_smokestate

This function writes value of s into the location of smokestate indexed by j.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
j	int	Standard
s	int	Standard

Table 2.3-23: set_smokestate Information.

2.3.2.1.12 get_smokestate

This function returns the contents of smokestate indexed by j.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
j	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
nomInfo -> equipTable[j].smokestate	int	the contents of smokestate indexed by j

Table 2.3-24: get_smokestate Information.

2.3.2.1.13 set_emstate

This function writes value of s into the location of emergency_offstate indexed by j.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
j	int	Standard
s	int	Standard

Table 2.3-25: set_emstate Information.

2.3.2.1.14 get_emstate

This function returns the contents of emergency_offstate indexed by j.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
j	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
nomInfo -> equipTable[j].emergency_offstate	int	the contents of emergency_offstate indexed by j

Table 2.3-26: get_em_state Information.

2.3.2.1.15 find_shmem_index

This function determines if la is equal to any of the values in lamp_addr. If so, it returns the index to location containing *la*. If not, this function returns -1.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
la	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
-1	int	search failed
j	int	the index to location containing *la*

Table 2.3-27: find_shmem_index Information

2.4 NOM-SIM Communications (Comms) CSC Description

The NOM-SIM Communications CSC (Comms) is distributed between processes resident on the NOM and SIM hosts. This CSC allows a technician to manage simulators remotely, from a central location at a NOM host console. It also allows the technician to control a single simulator from its locally attached SIM host console and tap into any management activity occurring at the NOM console that affects the simulator. Communications between NOM- and SIM- host-resident processes occur over the SIMNET network.

The Communications CSC enables access from NOM to a UNIX shell on any of the SIM hosts. This provides GUI with a remote SIM terminal window and MAP with a command pipe to send management and UNIX-like commands to a remote SIM host.UNIX shell. This CSC also enables NOM to start /stop a process on a remote SIM host and vice-versa; in either case, process output is viewable at a NOM console. This CSC also allows a console at the NOM host to display command input from a console attached to the remote host, and vice-versa. The Communications CSC has the following monitoring capability: it polls SIM hosts periodically for the presence or absence of Equipment PDUs. This serves as a heartbeat to indicate that a simulation process is running and connected to the SIMNET network.

The Communications CSC may be run without the rest of the NOM. The CSC is implemented as a distributed application, with a process on the NOM host and a process on each of the simulator hosts.

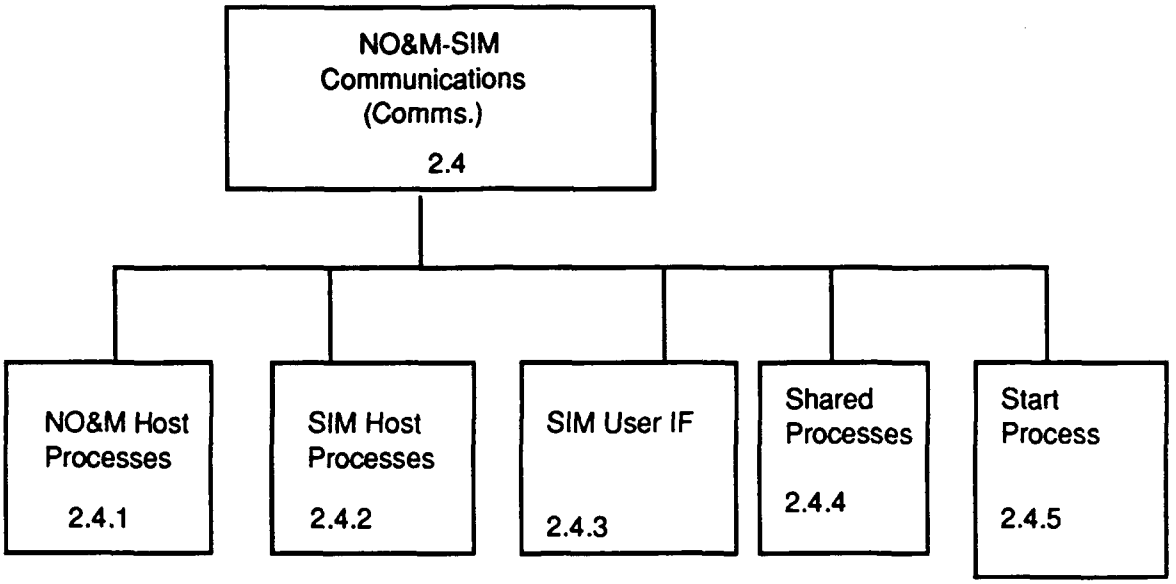
A Communications request protocol is primarily used within the Communications CSC. This request-response protocol is used to poll SIM hosts for a 'heartbeat.' The communications process resident on the NOM host also accepts requests from other NOM subsystems. GUI uses an attachMSGKind request indirectly to login and interact with remote simulators. The MAP uses the textMSGKind request to send commands to remote simulators.

The NOM Communications CSC is broken down into these lower-level CSCs:

- NOM Host Communications
- SIM Host Communications
- SIM User Interface
- Shared Processes
- Start a Process

SIM Host Communications CSC and SIM User Interface communicate with one another using a IPC, a local inter-process protocol; NOM Host Communications and SIM Host Communications communicate with one another using monitor protocol, which is designed for inter-process communications involving remote hosts.

Functional breakdown of the NOM-SIM Communications (Comms.) CSC is illustrated in Figure 2.4-1.



Note: IF=Interface

Figure 2.4-1: NOM-SIM Communications (Comms.) CSC Structure.

2.4.1 NOM Host Processes CSC

This lower-level CSC monitors each SIM host's 'heartbeat' to determine whether the latter is connected to the network and its processes are running. Such information is available in shared memory for access by MAP. This CSC can start/stop a process on a remote SIM host, request a remote SIM host UNIX shell, obtain a window into a remote SIM host UNIX shell for interacting with a SIM host shell from a NOM console, or for viewing interaction with a SIM host shell from a console locally attached to the SIM host. This CSC can also start/stop a remote SIM host shell session, and start or stop a process in a remote SIM host. This CSC also displays on a NOM screen output from a remote SIM host process. This CSC also records session history for all of the remote shells it has used.

Additionally, this CSC uses the CSU, mon_wind (Section 2.1.6.1).

Functional breakdown of this lower-level CSC is illustrated in Figure 2-4a.

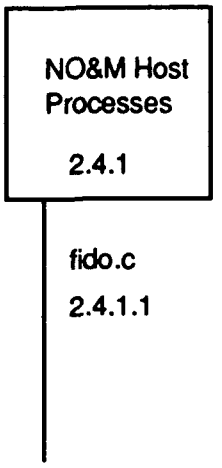


Figure 2.4-1:. Comms.--NOM Host Processes.

2.4.1.1 fido.c CSU Description (/simnet/cmd/nom/monitor)

This file contains source code for the monitor process that executes on the NOM host.

MachineStatus, ConnectionStatus are typedefined in this file.

This CSU runs on the NOM system host. It is invoked when the NOM system is started.

When mon_wind (Section 2.1.2.6) is used to resume access to an existing, remote shell, the fido process local to mon_wind may provide mon_wind with a transcript of the shell's recent output. This session history is recorded by fido for all the remote shells it has been involved with, up to some maximum number of shells and some maximum number of characters per shell.

This CSU optionally produces debugging output to stdout. It assumes the existence of a shared memory segment containing a table of simulators (placed there by other components of the NOM system)

2.4.1.1.1 main

This function is the program entry point. This function repeatedly processes input events, assembles a set of the file descriptors for accepting input on, listens for up to several seconds on those file descriptors, processes any input from the Ethernet, processes any input from window processes, checks whether it is time to poll each simulator out there, times out simulators that have not responded to polling, and closes connections with the failed machine.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
argc	int	Standard
argv	pointer to array of char	Standard

Calls	
Function	**Where Described**
AttachSharedMem	libipc in MCC CSCI
InitMonitorProcess	common.c
PollMonitor	this file
FatalSystemError	common.c
ProcessPacket	this file
ProcessWindowText	this file
CloseConnection	this file

Table 2.4-1: main Information.

2.4.1.1.2 ProcessPacket

This function reads and processes a packet from the Ethernet. It determines if the Monitor Response PDU came from a rover process and determines whether the simulator it represents is connected. If a Shell Status PDU reports that a connection has closed or failed to open, the routine discards the record of the connection. If sequenceNumber is zero, this function resynchronizes text sequence numbers.

Calls	
Function	**Where Described**
FatalSystemError	common.c
GET_DSAP	
GET_SSAP	
GET_CONTROL	
GET_PROTOID	
GET_ETHER_TYPE	
GET_DATA_PTR	
net_addr_bin_to_str	libnetif.a in MCC CSCI
SendMonitorResponsePDU	common.c
CloseSocket	libipc in MCC CSCI
YUMMCleanUp	libipc in MCC CSCI
LookupMachineByAddress	this file
ExecuteCommand	common.c
LookupConnection	this file
NotifyWindow	this file
CloseConnection	this file
CircleBufferCopyIn	
SendShellAckPDU	this file

Table 2.4-2: ProcessPacket Information.

2.4.1.1.3 PollMonitors

This function polls the monitors on simulators and other NOM systems.

Calls	
Function	**Where Described**
SendMonitorPDU	common.c

Table 2.4-3: PollMonitors Information.

2.4.1.1.4 SendShellOpenPDU

This function sends a Shell Open PDU.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
address	pointer to NetworkAddress	
channel	int	Standard
Calls		
Function	**Where Described**	
GET_DATA_PTR		
SendMonitorPDU	common.c	

Table 2.4-4: SendShellOpenPDU Information.

2.4.1.1.5 SendShellAckPDU

This function sends a Shell Ack PDU, acknowledging receipt of text.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
connect	pointer to register ConnectionStatus	
Calls		
Function	**Where Described**	
GET_DATA_PTR		
SendMonitorPDU	common.c	

Table 2.4-5: SendShellOpenPDU Information.

2.4.1.1.6 OpenConnection

This function open a new connection to a remote shell. It locates an unused connection table entry or reuses one that has no window process attached. If all entries are in use it tries to find an entry to reuse. This function initializes the new connection table entry.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
address	pointer to NetworkAddress	
channel	int	Standard
Calls		
Function	**Where Described**	
CircleBufferClear		
CircleBufferAlloc		

Table 2.4-6: OpenConnection Information.

2.4.1.1.7 CloseConnection

This terminates a connection when a remote shell exits.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
connect	pointer to ConnectionStatus	
Calls		
Function	**Where Described**	
net_addr_bin_to_str		
FatalSystemError	common.c	
CircleBufferFree		

Table 2.4-7: CloseConnection Information.

2.4.1.1.8 LookupConnection

This function finds a connection's status, given the remote machine's address and channel number.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
address	pointer to NetworkAddress	
channel	short	Standard
Return Values		
Return Value	**Type**	**Meaning**
connect	pointer to ConnectionStatus	Successful
0	pointer to ConnectionStatus	Unsuccessful
Calls		
Function	**Where Described**	
net_addr_compare		

Table 2.4-8: LookupConnection Information.

2.4.1.1.9 ProcessMessage

This function establishes communication with window process and other local clients. This function processes a YUMM message to kill a process, send a remote command, attach a window process to a remote shell, listen for output from a remote shell, and send shell text.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	int	Standard
msgLen	int	Standard
dummy	int	Standard
msgKind	long	Standard
msgData	pointer to register char	Standard
Calls		
Function	**Where Described**	
CloseSocket	libipc in MCC CSCI	
YUMMCleanUp	libipc in MCC CSCI	
SendRemoteCommandPDU	common.c	
FatalSystemError	common.c	
LookupConnection	this file	
CircleBufferAmount		
CircleBufferCopyOut		
OpenConnection	this file	
FatalSystemError	common.c	
SendShellOpenPDU	this file	
SetTerminalRaw	common.c	
LookupMachineByAddress	this file	
SendShellTextPDU	common.c	

Table 2.4-8: ProcessMessage Information.

2.4.1.1.10 ProcessWindowText

This function processes input from an window process. It reads some text from the window process' pseudo-tty. An EOF on the read signifies that the window is gone.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
connect	pointer to register ConnectionStatus	
Calls		
Function	**Where Described**	
FatalSystemError	common.c	
SendShellTextPDU	common.c	

Table 2.4-9: ProcessWindowText Information.

2.4.1.1.11 NotifyWindow

This sends a message string to a window process.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
connect	pointer to register ConnectionStatus	
str	pointer to char	Standard

Table 2.4-10: NotifyWindow Information.

2.4.1.1.12 LookupMachineByAddress

This function locates an entry in the NOM equipment table, given its Ethernet address.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
address	pointer to NetworkAddress	
Return Values		
Return Value	**Type**	**Meaning**
i	int	entry found
-1	int	entry not found
Calls		
Function	**Where Described**	
net_addr_compare		

Table 2.4-11: LookupMachineByAddress Information.

2.4.2 SIM Host Processes CSC Description

This lower-level CSC can create a remote UNIX shell on a SIM host and a command pipe to the shell for command input by NOM; obtain a window on a local SIM host UNIX shell for interacting with a SIM host shell from a console locally attached to the SIM host or for viewing NOM interaction with a SIM host UNIX shell. This CSC can also start/stop a process on NOM.

Functional breakdown of this lower-level CSC is illustrated in Figure 2.4.2-1.

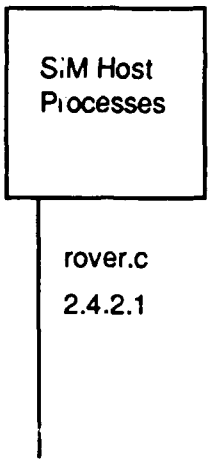


Figure 2.4.2-1: Comms.--SIM Host Processes.

2.4.2.1 rover.c CSU Description (/simnet/cmd/nom/monitor)

This file contains source code for the monitor process that executes on UNIX simulation hosts. ShellState is a typename typedefined in this file.

A rover process can spawn a shell and provide access to it (for typed input and displayed output) from either a local terminal or a remote NOM system. Several such spawned shells can exist at one time on a single machine; each is identified by a unique "channel number" (a low integer).

A shell spawned by a rover process can continue to exist even when no mon_lcmd, mon_wind, or tee_in process is being used to access it, and even when the rover process is no longer able to communicate with its companion fido process. Access to the shell can be resumed at any time.

This CSU optionally produces debugging output to stdout.

This CSU may include one or more channel numbers; at startup, rover will spawn a shell for each channel number specified.

2.4.2.1.1 main

This function is the program entry point. The routine does the following: initialization common to both monitor processes, uses the broadcast address for sending to the NOM system, catches signals reporting the death of spawned shells, spawns whatever shells are requested, assembles a set of the file descriptors for accepting input, accepts inupt from a shell it the input is not being conveyed to a remote system or it the input is being conveyed and there is buffer space remaining, accepts input from any tee_in process, listens for up to several seconds on those file descriptors, processes any input from the Ethernet, processes any input from the spawned shells and tee_in processes, and checks whether it is time to retry or abandon a transmission.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
argc	int	Standard
argv	pointer to array of char	Standard
Calls		
Function	**Where Described**	
FatalSystemError	common.c	
ProcessTeeText	this file	
ProcessShellText	this file	
CircleBufferAvail		
SpawnShell	this file	
InitMonitorProcess	common.c	

Table 2.4-12: main Information.

2.4.2.1.2 ProcessShellText

This function reads and forwards output generated by a spawned shell. If relaying text from the shell to a remote system, it accepts no more text than there is buffer capacity for. This function tries to read some text from a shell's PTY. If the shell has a remote client, this function transmits the text, saves a copy of it in the circular buffer, and forwards it to any tee-in process.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
shell	pointer to register ShellState	
Calls		
Function	**Where Described**	
CircleBufferAvail		
FatalSystemError	common.c	
SendShellStatusPDU	this file	
CloseShell	this file	
CircleBufferAmount		
CircleBufferCopyIn		

Table 2.4-13: ProcessShellText Information.

2.4.2.1.3 ProcessTeeText

This function reads and distributes text from a tee-in process.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
shell	pointer to register ShellState	
Calls		
Function	**Where Described**	
FatalSystemError	common.c	

Table 2.4-14: ProcessTeeText Information.

2.4.2.1.4 CatchCLDSignal

This function signals handler for SIGCLD signals, reporting that child processes have changed status.

Calls	
Function	**Where Described**
FatalSystemError	common.c

Table 2.4-15: CatchCLDSignal Information.

2.4.2.1.5 ProcessPacket

This function reads and processes a packet from the Ethernet. This function ensures that it is an 802.3 packet with the correct type code. This function interprets the packet as a Monitor PDU and checks its PDU version number.

This function performs these actions: responds to a Monitor Query with a Monitor Response, executes the command in a Remote Command PDU, spawns a new shell or reconnects to an existing one, notes the remote client to which this shell is connected, returns a PDU indicating success or failure, accepts text destined for a shell and sends it to the shell. If shell is not communicating, it is reset to communicate. This function also notes that receipt of text has been acknowledged.

Calls	
Function	**Where Described**
FatalSystemError	*common.c*
GET_DSAP	
GET_SSAP	
GET_CONTROL	
GET_PROTOID	
GET_ETHER_TYPE	
GET_DATA_PTR	
net_addr_bin_to_str	
SendMonitorResponsePDU	common.c
LookupShellByChannel	this file
SpawnShell	this file
CircleBufferClear	
SendShellStatusPDU	this file
CircleBufferAdvance	
CircleBufferEmpty	

Table 2.4-16: ProcessPacket Information.

2.4.2.1.6 RetryTextTransmission

This function retries or abandons transmission to a remote client.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
shell	pointer to register ShellState	
Calls		
Function	**Where Described**	
CircleBufferAmount		
CircleBufferCopyOut		
SendShellTextPDU	common.c	
SendShellStatusPDU	this file	

Table 2.4-17: RetryTextTransmission Information.

2.4.2.1.7 SendShellStatusPDU

This function sends a Shell Status PDU.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
address	pointer to NetworkAddress	
channel	int	Standard
status	ShellStatus	
Calls		
Function	**Where Described**	
GET_DATA_PTR		
SendMonitorPDU	common.c	

Table 2.4-18: SendShellStatusPDU Information.

2.4.2.1.8 SpawnShell

This function spawns and hooks up a new shell. This function attempts to find a free pseudo-tty and opens it for reading and writing. The master PTY's terminal characteristics are set.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
channel	short	Standard
Return Values		
Return Value	**Type**	**Meaning**
0	pointer to ShellState	unsuccessful
shell	pointer to ShellState	value of shell

Calls	
Function	**Where Described**
SetTerminalRaw	common.c
CircleBufferAlloc	
CloseShell	this file

Table 2.4-19: SpawnShell Information.

2.4.2.1.9 FUNCTION: void CloseShell (shell)

This function closes a shell.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
shell	pointer to register ShellState	
Calls		
Function	**Where Described**	
FatalSystemError	common.c	
CircleBufferFree		

Table 2.4-20: CloseShell Information.

2.4.2.1.10 LookupShellByChannel

This function looks up the shell by channel.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
channel	short	Standardd
Return Values		
Return Value	**Type**	**Meaning**
0	pointer to ShellState	shell not found
shell	pointer to ShellState	value of shell

Table 2.4-21: LookupShellByChannel Information.

2.4.2.1.11 FUNCTION: void ProcessMessage (type, msgKind, msgLen, msgData, dummy)

This function communicates with tee-in process and other local clients. It processes a YUMM message received from any of the clients. Messages received can be: kill a process, execute command on a remote system, open a shell connection to a tee-in process (a local client), or list open channels.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	int	Standard
msgLen	int	Standard
dummy	int	Standard
msgKind	long	Standard
msgData	pointer to register char	Standard
Calls		
Function	**Where Described**	
CloseSocket	libipc in MCC CSCI	
YUMMCleanUp	libipc in MCC CSCI	
SendRemoteCommandPDU	common.c	
FatalSystemError	common.c	
LookupShellByChannel	this file	
SetTerminalRaw	common.c	
TransmitConnectionList	this file	

Table 2.4-22: ProcessMessage Information.

2.4.2.1.12 TransmitConnectionList

This function sends a list of open shells to a tee-in process. It opens a pseudo-tty leading to the tee-in process, sends a text with the address of the NOM system, and sends a text with a list of channels open.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ptyName	pointer to char	Standard
Calls		
Function	**Where Described**	
FatalSystemError	common.c	
net_addr_bin_to_str		

Table 2.4-23: TransmitConnectionList Information.

2.4.3 SIM User Interface CSC Description

This lower-level CSC enables a user at a SIM console to tap into the interaction between a user at a NOM console and a SIM UNIX shell.

Functional breakdown of this lower-level CSC is illustrated in Figure 2.4.3-1.

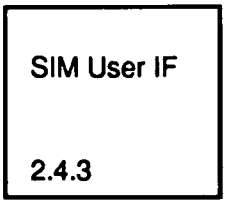


Figure 2.4.3-1: Comms.--SIM User Interface.

2.4.3.1 tee_in.c CSU Description (/simnet/cmd/nom/monitor)

This file contains source code for a program that taps in on connections maintained by rover, the remote system monitor process.

This CSU can be used to obtain a "window" into a shell spawned by a local rover process; it is used from a simulator host.

This CSU uses the channel number of the local shell. If this argument is not present, tee_in simply prints a list of the channel numbers of the local shells that currently exist, then tee_in exits. If the argument is present but it specifies a shell that does not exist, a shell is spawned for the specified channel.

Characters typed to tee_in's stdin are passed immediately to the local shell's stdin, without modification or local echoing (there are two exceptions noted below). Characters output by the local shell to its stdout or stderr are conveyed to tee_in and output to its stdout, also without modification. Tee_in recognizes two character sequences as special commands that it interprets rather than passing along to the local shell.

2.4.3.1.1 main

This function is the program entry point. This function calls functions to do the following: check number of command line arguments, look up the YUMM socket on which rover is listening, and open a free pseudo-tty for communicating with rover.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
argc	int	Standard
argv	pointer to array of char	Standard
Calls		
Function	**Where Described**	
YUMMInit	libipc in MCC CSCI	
YUMMProcess	libipc in MCC CSCI	
LkUpSocket	libipc in MCC CSCI	
ListConnections	this file	
AttachToChannel	this file	

Table 2.4-24: main Information.

2.4.3.1.2 ListConnections

This function obtains a list of active connections from rover.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fd	int	Standard
ptyName	pointer to char	Standard
Calls		
Function	**Where Described**	
FatalSystemError	common.c	
SendMsg	libipc in MCC CSCI	

Table 2.4-25: ListConnections Information.

2.4.3.1.3 AttachToChannel

This function taps into an open connection maintained by rover. A finite state machine is used to watch for escape sequences that terminate or suspend the tee_in process. NOM Comms is in normal state until a carriage return has been seen.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fd	int	Standard
channel	int	Standard
ptyName	pointer to char	Standard
Calls		
Function	**Where Described**	
SendMsg	libipc in MCC CSCI	
FatalSystemError	common.c	
SetTerminalRaw	common.c	
RestoreTerminal	common.c	

Table 2.4-26: AttachToChannel Information.

2.4.4 Shared Processes CSC Description

This lower-level CSC provides procedures and definitions for use by more than one NOM Communications CSC.

Functional breakdown of this lower-level CSC is illustrated in Figure 2.4.4-1.

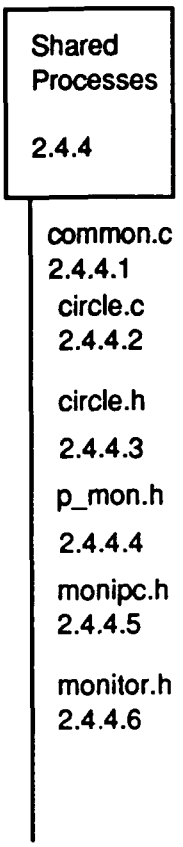


Figure 2.4.4-1: Comms.--Shared Processes.

2.4.4.1 common.c CSU Description (/simnet/cmd/nom/monitor)

This CSU contains an assortment of shared routines and global variables. This file contains common routines used by various monitor components.

2.4.4.1.1 InitMonitorProcess

This function does initialization common to both NOM host and SIM host monitor processes. This function obtains access to the Ethernet and initializes the use of YUMM. Before opening a YUMM socket, this function deletes any left-over ones having the same name. This function records process ID in /tmp/monitorpid.

Parameters		
Parameter	**Type**	**Where typedef Declared**
processMessage()	pointer to function returning void	Standard

Calls	
Function	**Where Described**
FatalSystemError	this file
YUMMInit	libipc in MCC CSCI
YUMMCleanUp	libipc in MCC CSCI
YUMMProcess	libipc in MCC CSCI
LkUpSocket	libipc in MCC CSCI
CloseSocket	libipc in MCC CSCI
OpenSocket	libipc in MCC CSCI
RecordProcessID	

Table 2.4-27: InitMonitorProcess Information.

2.4.4.1.2 FatalSystemError

This function prints a system error message in a window and exits.

2.4.4.1.3 ExecuteCommand

This function executes a system command.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
command	pointer to char	Standard

Table 2.4-28: ExecuteCommand Information.

2.4.4.1.4 FUNCTION: int OpenPseudoTty (ptyName)

This function finds and opens a pseudo-tty for communication with the monitor process. This function then sets the master pseudo-tty's terminal characteristics.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ptyName	pointer to char	Standard

Calls	
Function	**Where Described**
FatalSystemError	this file

Table 2.4-29: OpenPseudoTty Information.

2.4.4.1.5 SetTerminalRaw

This function sets a terminal's characteristics for raw, non-blocking input.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fd	int	Standard
Calls		
Function	**Where Described**	
FatalSystemError	this file	

Table 2.4-30: SetTerminalRaw Information.

2.4.4.1.6 FUNCTION: void RestoreTerminal (fd)

This function restores terminal's characteristics to normal.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fd	int	Standard
Calls		
Function	**Where Described**	
FatalSystemError	this file	

Table 2.4-31: RestoreTerminal Information.

2.4.4.1.7 SendMonitorResponsePDU

This function sends a Monitor Response PDU.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
address	pointer to NetworkAddress	
kind	MonitorKind	
Calls		
Function	**Where Described**	
GET_DATA_PTR		
SendMonitorPDU	this file	

Table 2.4-32: SendMonitorResponsePDU Information.

2.4.4.1.8 SendRemoteCommandPDU

This function sends a Remote Command PDU.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
address	pointer to NetworkAddress	
length	int	Standard
text	pointer to char	Standard
Calls		
Function	**Where Described**	
GET_DATA_PTR		
SendMonitorPDU	this file	

Table 2.4-33: SendRemoteCommandPDU Information.

2.4.4.1.9 FUNCTION: void SendShellTextPDU (address, channel, sequenceNumber, buf, length)

This function sends a Shell Text PDU.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
address	pointer to NeworkAddress	
channel	int	Standard
length	int	Standard
sequenceNumber	long	Standard
buf	pointer to char	Standard
Calls		
Function	**Where Described**	
GET_DATA_PTR		
SendMonitorPDU	this file	

Table 2.4-34: SendShellTextPDU Information.

2.4.4.1.10 SendMonitorPDU

This function sends a Monitor PDU via /dev/enp1. The Ethernet packet's header is filled in and the packet is transmitted.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
address	pointer to NetworkAddress	
pkt	pointer to NetworkPacket	
length	int	Standard
kind	int	Standard

Calls	
Function	**Where Described**
FatalSystemError	this file
net_addr_bin_to_str	
SET_DSAP	
SET_SSAP	
SET_CONTROL	
SET_PROTOID	
SET_ETHER_TYPE	

Table 2.4-35: SendMonitorPDU Information.

2.4.4.2 circle.c CSU Description (/simnet/cmd/nom/monitor)

This CSU contains routines implementing circular text buffers.

2.4.4.3 circle.h CSU Description (/simnet/cmd/nom/monitor)

This CSU contains definitions associated with the routines in circle.c.

2.4.4.4 p_mon.h CSU Description (/simnet/cmd/nom/monitor)

This file defines the representation of Monitor Protocol Data Units. These include PDUs that query for the presence of monitors on the network; PDUS for executing a remote command; and PDUs for communicating with a remote shell.

This CSU is (derived from /simnet/lib/protocol/p_mon.drn) and defines the monitor protocol used by fido and rover to communicate.

Rover and fido processes communicate via the simulation Ethernet using type code 21001 (decimal). The protocol used is the "monitor protocol", defined in /simnet/lib/protocol/p_mon.drn. Monitor PDUs are placed in Ethernet packets with IEEE 802.3 headers.

The monitor protocol provides a flow-controlled text stream from a shell running under a rover process to a fido process. A positive acknowledgement scheme is used: fido sends an acknowledgement to rover each time text is successfully received. Character sequence numbers are used to positions in the text stream for the purpose of acknowledgement. If rover fails to receive an acknowledgement for text it has sent, it will retry the transmission at intervals up to some maximum number of times.

Flow control and acknowledgements are used for text transmitted in one direction only: from simulator to NOM system. Text from a NOM system to a simulator is transmitted once and once only.

2.4.4.5 monipc.h CSU Description (/simnet/cmd/nom/include)

This file defines YUMM messages communicated between a monitor process (fido or rover) and its clients.

This CSU defines two forms of local IPC used among processes making up the monitor system:

- YUMM is used to catch the attention of a monitor process (fido or rover) and make a request of it

- pseudo-tty's are used to pass text between processes

Uses of IPC:

IPC is used by mon_wind is used to spawn a remote shell and provide access to it. Initially, fido is running on a NOM host and rover is running on a simulator host.

When mon_wind is invoked on the NOM host, it locates both a free pseudo-tty and the YUMM socket of its local fido process. It opens the pseudo-tty for reading and writing text, and sends a YUMM message to fido's socket that identifies that pseudo-tty and requests access to a remote shell through it. Upon receiving the YUMM message from mon_wind, fido sends a PDU to the rover process running on the simulator host. Upon receiving the PDU, rover locates a free pseudo-tty, forks a shell, and uses the pseudo-tty to read and write stdout, stderr, and stdin of the forked shell.

While the session is open, mon_wind, fido, rover, and the shell simply pass text among each other using pseudo-ttys. When the shell terminates, rover finds out by reading an EOF on its end of the pseudo-tty shared with the shell. Rover sends a PDU to fido notifying it that the shell has quit. When fido receives this PDU, it closes its end of the pseudo-tty shared with mon_wind. As a result, mon_wind reads an EOF on its end of that pseudo-tty and it quits.

2.4.4.6 monitor.h CSU Description (/simnet/cmd/nom/monitor)

This CSU contains declarations common to NOM Comms CSUs. It contains definitions shared among monitor components.

2.4.5 Start a Process CSC Description

This lower-level CSC enables a user at a NOM or SIM console to start a remote process at the other console's host.

This CSC also enables a user at a SIM console to start simulation application software from a local SIM UNIX shell. This CSC includes a script, autostart, which is used to automatically start the M1 simulation and tee-in processes..

Functional breakdown of this lower-level CSC is illustrated in Figure 2.4.5-1.

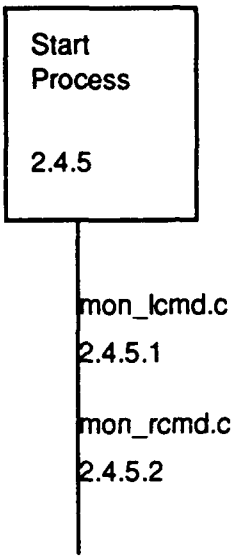


Figure 2.4.5-1: Comms.--Start SIM Application.

2.4.5.1 mon_lcmd.c CSU Description (/simnet/cmd/nom/monitor)

This CSU can be used on a simulator host to cause a shell to be spawned by its rover process; it can also be used to pass a command to a spawned shell.

This program executes a command within a spawned shell on the local system via the local rover.

This CSU uses the channel number of the shell to be accessed; if that shell doesn't exist, it is spawned. This CSU passes an optional command to the spawned shell. The command may be composed of multiple words. Before being passed to the shell the words will be concatenated with intervening blanks so that commands like, for example, "mon_lcmd 200 cflf303801 reboot -h" will have the desired effect.

2.4.5.1.1 int main (argc,argv)

This function is the program entry point.

This function does the following: checks command line arguments, assembles a command by concatenating command line arguments, locates the rover process, opens a free pseudo-tty for communicating with fido, sends the command to rover, and closes the connection with rover.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
argc	int	Standard
argv	pointer to array of char	Standard

Calls	
Function	**Where Described**
YUMMInit	libipc in MCC CSCI
YUMMProcess	libipc in MCC CSCI
LkUpSocket	libipc in MCC CSCI
OpenPseudoTty	common.c
SendMsg	libipc in MCC CSCI
FatalSystemError	common.c
YUMMCleanUp	libipc in MCC CSCI

Table 2.4-36: main Information.

2.4.5.2 *mon_rcmd.c CSU Description (/simnet/cmd/nom/monitor)*

This program executes a command on a remote system via local and remote monitor processes.

This CSU can be used on either a NOM or a simulator host to execute a program on a remote system.

Mon_rcmd invokes a command on a remote system via local and remote monitor processes. The local and remote systems may each be running either fido or rover. The command is invoked on the remote system by a system(2) call. The command may be composed of multiple words. Before being passed to the remote system the words will be concatenated with intervening blanks so that commands will have the desired effect.

This CSU uses the simulation Ethernet address of the remote system.

2.4.5.2.1 main

This function is the program entry point.

This function does the following: checks and concatenates command line arguments, locates the local monitor process and sends it a remote command message.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
argc	int	Standard
argv	pointer to array of char	Standard
Calls		
Function	**Where Described**	
YUMMInit	libipc in MCC CSCI	
YUMMProcess	libipc in MCC CSCI	
LkUpSocket	libipc in MCC CSCI	
net_addr_str_to_bin		
SendMsg	libipc in MCC CSCI	
FatalSystemError	common.c	
YUMMCleanUp	libipc in MCC CSCI	

Table 2.4-37: main Information.

2.5 NOM-Level Processes CSC Description

This CSC provides NOM initialization procedures and interface definitions for use by more than one NOM top-level CSC.

Functional breakdown of NOM-Level Processes CSC is illustrated in Figure 2.5-1.

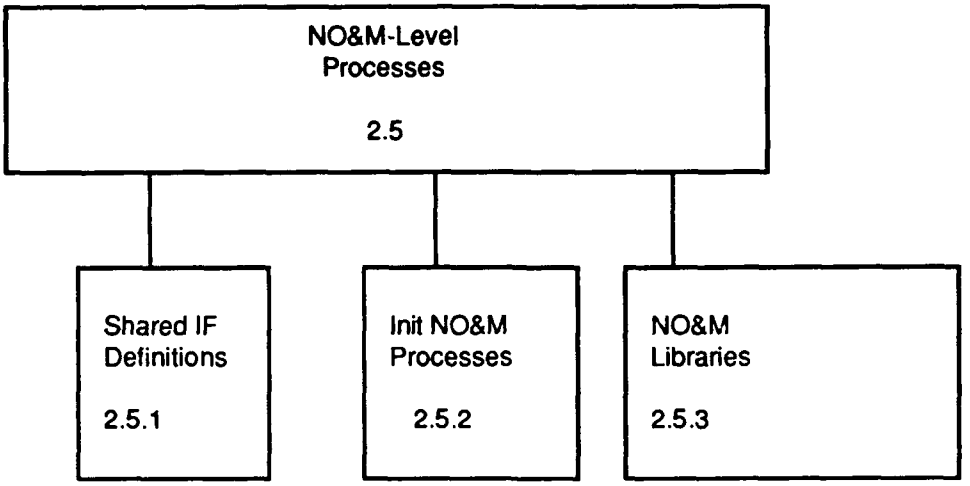


Figure 2.5-1: NOM-Level Processes CSC Structure.

2.5.1 Shared Interface Definitions CSC Description

This lower-level CSC contains protocol definitions used by several NOM CSCs.
A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.5.1-1.

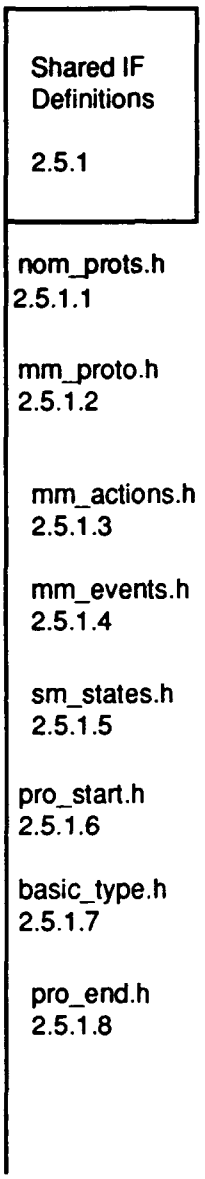


Figure 2.5.1-1: NOM Top-Level Processes--Shared Interface Definitions.

2.5.1.1 nom_prots.h CSU Description (/simnet/cmd/nom/include)

This CSU defines protocols use by NOM.

2.5.1.2 mm_proto.h CSU Description (/simnet/cmd/nom/include)

This CSU defines format for Manager-Manager protocol.

2.5.1.3 mm_actions.h CSU Description (/simnet/cmd/nom/include)

This CSU defines enumerated values of action IDs, and action response codes. Action IDs are used for requesting operations to be performed on managed resources, (e.g, rebooting a simulator) via a mmpActionReqPDU. These definitions conform to the formats defined in mm_proto.h (Section 2.5.1.2).

2.5.1.4 mm_events.h CSU Description (/simnet/cmd/nom/include)

This CSU defines enumerated values of event IDs. This file contains registered event IDs, used for reporting errors and other occurrences recorded in an mmpEventPDU. These definitions conform to the formats defined in mm_proto.h (Section 2.5.1.2).

2.5.1.5 sm_states.h CSU Description (/simnet/cmd/nom/include)

This CSU contains enumerated values for equipment state attributes maintained by different NOM processes.

2.5.1.6 pro_start.h CSU Description (/simnet/cmd/nom/include)

This CSU contains definitions that precede includes of protocol header files. It defines macros which are used to eliminate naming conflicts with names chosen in protocol header files.

2.5.1.7 basic_type.h CSU Description (/simnet/cmd/nom/include)

This CSU defines commonly-used basic types and extensions.

2.5.1.8 pro_end.h CSU Description (/simnet/cmd/nom/include)

This CSU contains definitions that follow includes of protocol header files. It undefines macros which are used to eliminate naming conficts with names chosen in protocol header files.

2.5.2 Initialize NOM Processes CSC Description

This lower-level CSC contains scripts for starting/stopping NOM processes.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.5.2-1.

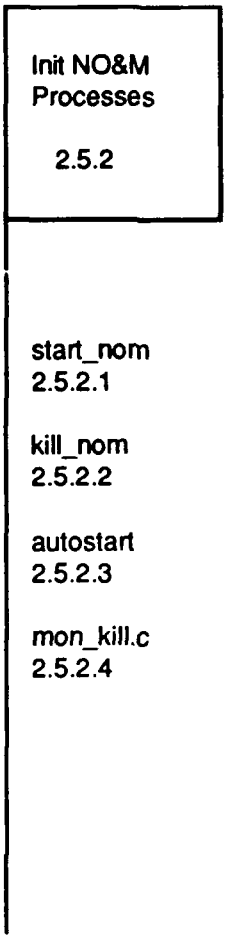


Figure 2.5.2-1: NOM Top-Level Processes--Initialize NOM Processes.

2.5.2.1 start_nom CSU Description (/simnet/cmd/nom/scripts)

NOM is started by the script nom/script/start_nom. This script runs nom/bin/nom_netstart, which loads nom_enp.bin into the CMC card, and then starts the NOM processes, gui, mini (MAP), lamp, and monitors NOM host program entry point.

2.5.2.2 kill_nom CSU Description (simnet/cmd/nom/scripts)

This CSU is used to kill all previous NOM processes and clean up the state of NOM before it is restarted.

2.5.2.3 autostart CSU Description (simnet/cmd/nom/scripts)

This CSU resides on actual simulators. It is used to automatically start monitor processes on a simulator.

2.5.2.4 nom_netstart.c CSU Description (simnet/cmd/nom/monitor)

This CSU loads the CMC card with a NOM-specific filter at startup time.

2.5.3 NOM Libraries CSC Description

This lower-level CSC contains libraries whose functions are called by NOM CSCs. These libraries are SIMNET-wide.

A functional breakdown of this lower-level CSC into CSUs is shown in Figure 2.5.3-1.

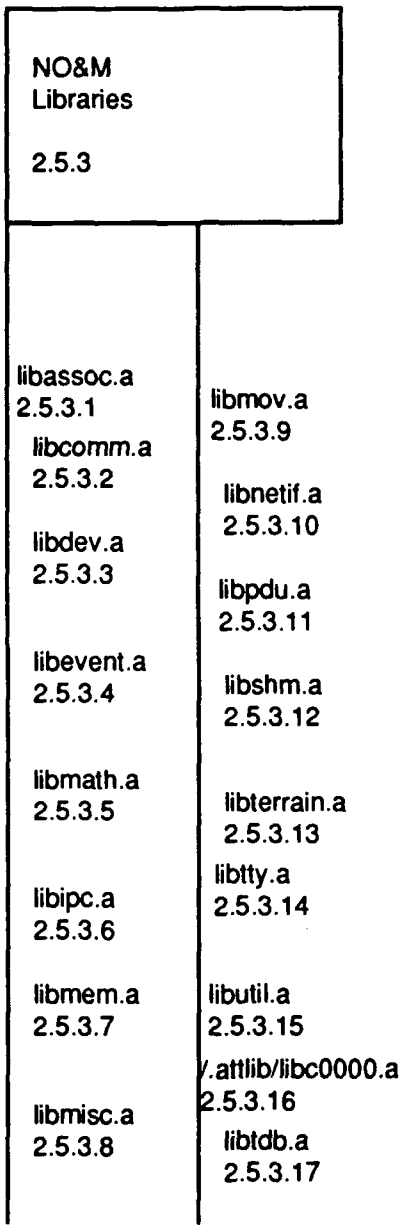


Figure 2.5.3-1: NOM Top-Level Processes--NOM Libraries.

2.5.3.1 libassoc

This CSU is a SIMNET system library. It provides functions for the association layer of the SIMNET protocols. This library is documented in Section 2.20.1 of the MCC CSCI document.

2.5.3.2 libcomm (/simnet/lib)

This CSU is a SIMNET system library.

2.5.3.3 libdev CSU Description (/simnet/lib)

This CSU is a SIMNET system library.

2.5.3.4 libevent CSU Description (/simnet/lib)

This CSU is a SIMNET system library documented under the Vehicles CSCI.

2.5.3.5 libmath CSU Description (/simnet/lib)

This CSU is a SIMNET system library documented under the Vehicles CSCI

2.5.3.6 libipc CSU Description (/simnet/lib)

This CSU is a SIMNET system library documented under the MCC CSCI. It contains the YUMM library whose functions are used by NOM for IPC message-passing.

2.5.3.7 libmem CSU Description (/simnet/lib)

This CSU is a SIMNET system library documented under the Vehicles CSCI.

2.5.3.8 libmisc CSU Description (/simnet/lib)

This CSU is a SIMNET system library.

2.5.3.9 libmove CSU Description (/simnet/lib)

This CSU is a SIMNET system library documented under the Vehicles CSCI.

2.5.3.10 libnetif CSU Description (/simnet/lib)

This CSU is a SIMNET system library documented under the MCC CSCI.

2.5.3.11 libpdu CSU Description (/simnet/lib)

This CSU is a SIMNET system library. It is used by NOM for defining and passing PDUs.

2.5.3.12 libshm CSU Description (/simnet/lib)

This CSU is a SIMNET system library documented under the Vehicles CSCI. It is used by NOM for shared memory access.

2.5.3.13 libterrain CSU Description (/simnet/lib)

This CSU is a SIMNET system library.

2.5.3.14 libtty CSU Description (/simnet/lib)

This CSU is a SIMNET system library.

2.5.3.15 libutil CSU Description (/simnet/lib)

This CSU is a SIMNET system library documented under the Vehicles CSCI.

2.5.3.16 /attlib/libc0000.a

This CSU is a standard "C" library. It is used for graphics applications.

2.5.3.17 libtdb

This CSU is a SIMNET system library containing terrain database functions documented under the MCC CSCI.

APPENDIX A: TYPE NAMES AND WHERE THEY ARE DEFINED

Type Names	Where Defined
ScreenClassRec;	screens.h
ActionRsp;	mm_actions.h
AttachMessage;	monipc.h
CircleBuffer;	circle.h
ErrorCode;	global.h
Event;	events.h
f_text	text.h
FieldRec;	screens.h
ListRequestMessage;	monipc.h
MMP_PDUData;	mm_proto.h
MMP_PDUStruct;	mm_proto.h
NomEquipInfo;	nom_shmem.h
NomInfo;	nom_shmem.h
pr_size	xglobals.h
PR_SIZE;	xglobals.h
RemoteCommandMessage;	monipc.h
ScreenRec;	screens.h
ScreenWidget;	screens.h
ScreenWidgetClass;	screens.h
SP_Monitor_PDU;	p_mon.h
SP_Monitor_PDUKind;	p_mon.h
SP_MonitorKind;	p_mon.h
SP_MonitorProtocol Version;	p_mon.h
SP_MonitorResponse Variant;	p_mon.h
SP_RemoteCommand Variant;	p_mon.h
SP_ShellAckVariant;	p_mon.h
SP_ShellOpenVariant;	p_mon.h
SP_ShellStatus;	p_mon.h
SP_ShellStatusVariant;	p_mon.h
SP_ShellTextVariant;	p_mon.h
switch_struct	panel.h
WriteMessage;	monipc.h

APPENDIX B: MACRO FUNCTION NAMES AND WHERE THEY ARE DEFINED

Macro Function Names	Where Defined
char_advance(font,char)	fonts.h:
char_height(font)	fonts.h:
char_width(font)	fonts.h:
CircleBufferAmount(buf)	circle.h:
CircleBufferAvail(buf)	circle.h:
CircleBufferClear(buf)	circle.h:
CircleBufferEmpty(buf)	circle.h:
FillInNetHdr(b, a)	MCC_net.h:
ICON_NAME_X(i)	icons.c
ICON_NAME_Y(i)	icons.c
ICON_STATE_X(i)	icons.c
ICON_STATE_Y(i)	icons.c
NetTableEntry(n)	MCC_net.h:
NetTableUnlock(n)	MCC_net.h:

INDEX BY SECTION NUMBER