(1)

SOFTWARE DESIGN DOCUMENT
Vehicle Simulation CSCI (5)

Volume 3 of 4 Sections 2.5.4 - 2.6.18.12.1

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

SOFTWARE DESIGN DOCUMENT
Vehicle Simulation CSCI (5)

Volume 3 of 4 Sections 2.5.4 - 2.6.18.12.1

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

SOFTWARE DESIGN DOCUMENT
Vehicle Simulation CSCI (5)

Volume 3 of 4 Sections 2.5.4 - 2.6.18.12.1

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
June 1991

3. REPORT TYPE AND DATES COVERED
Software Design Document

4. TITLE AND SUBTITLE
Software Design Document Vehicle Simulation CSCI (5)

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 #:
9108

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 Vehicle Simuation Computer Software Configuration Item (CSCI number 5) of the SIMNET hardware and software training system for vehicle crew training and operational training.

14. SUBJECT TERMS
SIMNET Software Design Document for the Vehicle Simulation CSCI (CSCI 5).

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT
Same as report.

2.5.4 libfail
(./simnet/release/src/vehicle/libsrc/libfail [libfail])

This library determines the damages to the vehicle due to failures. Failures are divided into categories that indicate the method used for failure generation. The three categories are combat damage, stochastic failure, and deterministic failure:

1) During combat a vehicle receives combat information messages from the network. This information comes in two different forms. First, impact message tells the vehicle that someone has been hit by an incoming direct fire round or missile (both referred to as a round). If the round struck another vehicle, then the message is ignored for purposes of combat damage. The vehicle struck by the round uses the information in the message to calculate any damages that may result. Second, an indirect fire message tells the vehicle that an indirect fire round has exploded. The impact point is checked to determine if the impact was close enough to damage the vehicle. The combat damage tables are read in through one file which contains references to other data files.

2) Deterministic failures are those failures which result from some improper action by the crew that generally could have been prevented. These include both failures due to resource depletion and failures due to crew error. Examples of these errors include mismanaging fuel and ammunition, ignoring warning lights, and throwing a track while driving the vehicle across a hill with too great a slope. The deterministic failure table is read in through a single data file.

3) A stochastic failure occurs when a vehicle fails on its own and not because of a crew error or due to combat damage. The frequency of failure is determined by a Mean Number of Operations Between Failures (MNOBF). Stochastic failures can degrade functions or can serve as a warning for potential deterministic failures. Stochastic failures are determined with the use of a data file (nal_sdamage.d).

2.5.4.1 c_chk_dam.c

(./simnet/release/src/vehicle/libsrc/libfail/c_chk_dam.c)

This file contains the code to generate combat damage. An index is generated from hit information and is used to test for combat kill or damage.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"pro_data.h"
"status.h"
"fail.h"
"fail_loc.h"
"cfail_loc.h"
"libnetwork.h"

2.5.4.1.1 cfail_check_damages

This routine is used by failures to check the combat damage list.

damage_list -- a pointer to a linked list of possible damages to the vehicle
agent_id, event_id, cause -- passed to the network if sending a StatusChangePDU.

If the vehicle is destroyed, the PDU will be sent from the routine **fail_vehicle_is_destroyed**(), rather than from **cfail_check_damages**(). Note that *rtn_index* of -1 indicates catastrophic kill. For debugging purposes only, the vehicle will not die when *CFAIL_DEBUG* has been turned on (except in special circumstances such as the blast door is open).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
damage_list	pointer to register FAIL_TEST	Section 2.5.4.15
agent_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
event_id	EventID	/simnet/common/include/protocol/basic.h
cause	DamageCause	/simnet/common/include/protocol/p_data.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
damage_prob	register REAL	/simnet/common/include/global/sim_types.h
percent	REAL	/simnet/common/include/global/sim_types.h
subsys	pointer to VehicleSubsystems	/simnet/common/include/global/status.h

Errors	
Error	**Reason for Error**
stderr	- Checking damages - possible damage = #, prob = #, range = %
Calls	
Function	**Where Described**
rand	Section 2.5.4.18
fail_vehicle_is_destroyed	Section 2.5.4.9.2
fail_get_delta_subsystems	Section 2.5.4.14.8
network_send_status_change	Section 2.1.1.3.1.62.1

Table 2.5-97: cfail_check_damages Information.

2.5.4.2 c_debug.c

(./simnet/release/src/vehicle/libsrc/libfail/c_debug.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"

Declarations:

CFAIL_DEBUG

2.5.4.2.1 cfail_debug_on

This routine is used for debugging purposes.

2.5.4.3 c_dir_fire.c

(./simnet/release/src/vehicle/libsrc/libfail/c_dir_fire.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libmap.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"fail.h"
"cfail_loc.h"
"libfail.h"
"libnetwork.h"
"pro_sim.h"
"pro_data.h"

Defines:

PI_DIV_6
PI_DIV_3
PI_DIV_2
FIVE_PI_DIV_6
TWO_PI_DIV_3
RATIO_LENGTH_TO_WIDTH

Variable and Procedure Declarations:

c_dir_fire_debug
cfail_compute_impact_incidence_angle()
normalize_x()
normalize_y()

2.5.4.3.1 cfail_dir_fire_damages

This routine is called whenever it is necessary to compute direct fire damages. The parameter *hit_message* contains information about the hit (shell type and hit location). This routine calls the necessary routines to check for both catastrophic kill and normal combat damages. Note that this routine will not be called under certain circumstances, for example, if the M1 is hit by a large calibre round when the blast door is open, "m1_failure.c" will skip this routine and make a call directly to **fail_cat_kill**().

Parameters		
Parameter	**Type**	**Where Typedef Declared**
hit_msg	pointer to register *ImpactVariant*	/simnet/common/include/protocol/p_sim.h
ammo_type	register int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
hit_loc	register int	Standard
this_ammo	pointer to register GEN_AMMO_DAMAGES	Section 2.5.4.7
damage_file_index	int	Standard

Errors	
Error	**Reason for Error**
stderr	- CFAIL: invalid ammo type - no damage table for ammo type yet - can't do dir fire damage on indirect ammo_type - composite hit location - invalid dam_info for ammo type

Calls	
Function	**Where Described**
map_get_damage_file_index_from_ammo_entry	Section 2.6.11.2.12
cfail_get_composite_index	Section 2.5.4.3.2
cfail_check_damages	Section 2.5.4.1.1

Table 2.5-98: cfail_dir_fire_damages Information.

2.5.4.3.2 cfail_get_composite_index

This routine uses the vehicle impact packet's account of where the incoming round hit (the vehicle location, the side of the vehicle that was hit, and the incidence angle) to create a single composite hit location which is used as an index into the damage table.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
hit_msg	pointer to register ImpactVariant	/simnet/common/include/protocol/p_sim.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
hit_location	int	Standard
incidence_angle	double	Standard
Return Values		
Return Value	**Type**	**Meaning**
hit_location	int	composite hit location; index into damage table
Calls		
Function	**Where Described**	
cfail_compute_impact_incidence_angle	Section 2.5.4.3.3	
cfail_compute_side_hit	Section 2.5.4.3.4	

Table 2.5-99: cfail_get_composite_index Information.

2.5.4.3.3 cfail_compute_impact_incidence_angle

This routine computes the incidence angle based on the trajectory and vehicle coordinates. Be aware that this code defines an incidence angle as an angle measured horizontally on the horizon. Certain military combat damage modelers define an "incidence angle" as an angle measured vertically above the horizon and an "aspect angle" as an angle measured horizontally on the horizon.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
trajectory[]	float	Standard
Return Values		
Return Value	**Type**	**Meaning**
trajectory[0], trajectory[1]	double	the incidence angle (measured in radians)

Table 2.5-100: cfail_compute_impact_incidence_angle Information.

2.5.4.3.4 cfail_compute_side_hit

This routine computes the side hit, determining in which quadrant of the vehicle the impact occurred based on impact information and the incidence angle.

2	1
3	4

Parameters		
Parameter	**Type**	**Where Typedef Declared**
impact[]	float	Standard
incidence_angle	double	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
temp_x	double	Standard
temp_y	double	Standard
hit_location	int	Standard

Return Values		
Return Value	**Type**	**Meaning**
hit_location	int	index into damage table

Calls	
Function	**Where Described**
compute_incidence_from_front	Section 2.5.4.3.8
compute_incidence_from_right	Section 2.5.4.3.10
normalize_x	Section 2.5.4.3.5
normalize_y	Section 2.5.4.3.6
compute_incidence_from_left	Section 2.5.4.3.9
compute_incidence_from_back	Section 2.5.4.3.7

Table 2.5-101: cfail_compute_side_hit Information.

2.5.4.3.5 normalize_x

This routine normalizes the x dimension of the vehicle size to between 0 and 1.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
x_pos	float	Standard

Return Values		
Return Value	**Type**	**Meaning**
x_pos	float	the normalized dimension (between 0 and1)

Table 2.5-102: normalize_x Information.

2.5.4.3.6 normalize_y

This routine normalizes the y dimension of the vehicle size to between 0 and 1.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
y_pos	float	Standard
Return Values		
Return Value	**Type**	**Meaning**
y_pos / RATIO_LENGTH_TO_WIDTH	float	the normalized x dimension (between 0 and 1)

Table 2.5-103: normalize_y Information.

2.5.4.3.7 compute_incidence_from_back

This routine categorizes the incidence angle from the back of the vehicle into either 0-30, 30-60, or 60-90 degrees.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
incidence_angle	double	Standard
Return Values		
Return Value	**Type**	**Meaning**
INCIDENCE_0_30	int	the incidence angle is between 0 and 30 degrees
INCIDENCE_30_60	int	the incidence angle is between 30 and 60 degrees
INCIDENCE_60_90	int	the incidence angle is between 60 and 90 degrees

Table 2.5-104: compute_incidence_from_back Information.

2.5.4.3.8 compute_incidence_from_front

This routine categorizes the incidence angle from the front of the vehicle into either 0-30, 30-60, or 60-90 degrees.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
incidence_angle	double	Standard
Return Values		
Return Value	**Type**	**Meaning**
INCIDENCE_0_30	int	the incidence angle is between 0 and 30 degrees
INCIDENCE_30_60	int	the incidence angle is between 30 and 60 degrees
INCIDENCE_60_90	int	the incidence angle is between 60 and 90 degrees

Table 2.5-105: compute_incidence_from_front Information.

2.5.4.3.9 compute_incidence_from_left

This routine categorizes the incidence angle from the left side of the vehicle into either 0-30, 30-60, or 60-90 degrees.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
incidence_angle	double	Standard
Return Values		
Return Value	**Type**	**Meaning**
compute_incidence_from_back(incidence_angle)	int	the category of the incidence angle: 0-30, 30-60, 60-90
Calls		
Function	**Where Described**	
compute_incidence_from_back	Section 2.5.4.3.7	

Table 2.5-106: compute_incidence_from_left Information.

2.5.4.3.10 compute_incidence_from_right

This routine categorizes the incidence angle from the right side of the vehicle into either 0-30, 30-60, or 60-90 degrees.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
incidence_angle	double	Standard
Return Values		
Return Value	**Type**	**Meaning**
compute_incidence_from_back(incidence_angle)	int	the category of the incidence angle: 0-30, 30-60, 60-90
Calls		
Function	**Where Described**	
compute_incidence_from_back	Section 2.5.4.3.7	

Table 2.5-107: compute_incidence_from_right Information.

2.5.4.4 c_ind_fire.c
(./simnet/release/src/vehicle/libsrc/libfail/c_ind_fire.c)

This file is used to determine damage and failures resulting from indirect fire. Indirect fire explodes adjacent to a vehicle rather than directly on a vehicle.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"basic.h"
"pro_sim.h"
"pro_data.h"
"mun_type.h"
"fail.h"
"cfail_loc.h"
"libmatrix.h"
"libnetwork.h"
"libmap.h"
"libfail.h"

Variable Declaration:

c_ind_fire_debug

2.5.4.4.1 cfail_indirect_fire_damages

This routine determines the damages to the vehicle from indirect fire. The index to the indirect fire damage table is calculated from the ammo type, the distance of the explosion from the vehicle, and the side of the vehicle that the explosion occurred on. *ammo_type* is the index to array of ammunition structures in libmap.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_type	int	Standard
detonator	ObjectType	/simnet/common/include/protocol/p_sim.h
shot	pointer to IndirectFireDetonation	/simnet/common/include/protocol/p_sim.h
range_sqrd	REAL	/simnet/common/include/global/sim_types.h
h_to_o	VECTOR	/simnet/common/include/global/sim_types.h
w_to_h	T_MATRIX	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
hit_pos	VECTOR	/simnet/common/include/global/sim_types.h
hit_loc	int	Standard
this_ammo	pointer to GEN_AMMO_DAMAGES	Section 2.5.4.7
damage_file_index	int	Standard

Errors	
Error	**Reason for Error**
stderr	- checking ind fire: range_sqrd = #, ammo = # - CFAIL: invalid ammo type - invalid dam_info for amotype

Calls	
Function	**Where Described**
map_get_damage_file_index_from_ammo_entry	Section 2.6.11.2.12
vec_mat_mul	Section 2.6.2.56.1
vec_sub	Section 2.6.2.64.1
cfail_get_indirect_index	Section 2.5.4.4.2
cfail_check_damages	Section 2.5.4.1.1

Table 2.5-108: cfail_indirect_fire_damages Information.

2.5.4.4.2 cfail_get_indirect_index

This routine determines the index to the indirect fire damage table and returns the composite hit location, which is used as the index to the damage table. The index to the indirect fire damage table is calculated from the ammo type, the distance of the explosion from the vehicle, and the side of the vehicle that the explosion occurred on.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo	int	Standard
detonator	ObjectType	/simnet/common/include/protocol/p_sim.h
range_sqrd	REAL	/simnet/common/include/global/sim_types.h
ranges	pointer to REAL	/simnet/common/include/global/sim_types.h
hit_pos	pointer to REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
hit_location	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
hit_location	int	the composite hit location used as the index to the indirect fire damage table

Table 2.5-109: cfail_get_indirect_index Information.

2.5.4.5 c_init.c
(./simnet/release/src/vehicle/libsrc/libfail/c_init.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"fail.h"
"cfail_loc.h"
"libfail.h"
"simstdio.h"

Procedure and Variable Declarations:

cfail_cdamages_init()
cfail_kill_init()
malloc()
exit() --Simnet Butterfly Machine only
free()
init_indirect_fire_table()
init_direct_fire_table()
damage_file_root[50]

Defines:

COMMENT_SIZE

2.5.4.5.1 cfail_init

This routine initializes the combat failures module. The routine takes one parameter, *cd_file_root*, the directory path used to search for the combat damage file , for example, "/simnet/vehicle/m1/data/m1". The combat damage file is read in through **cfail_read_damage_file()**. For each type of ammo, the damage_by_type array contains information about each ammo type and how the vehicle can be damaged by that ammo type. This routine also zeroes out the damage_by_type array, in order to ignore unkown ammo types.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
cd_file_root	pointer to char	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard

Table 2.5-110: cfail_init Information.

2.5.4.5.2 cfail_read_damage_file

This routine initializes the combat damage table for the specified type of ammunition. After formatting the damage file name, it makes sure that the file can be opened, and then calls another routine actually read it in, based on whether the ammo is direct or indirect fire. This routine returns TRUE if the file was opened sucessfully, and FALSE otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
damage_file_suffix	pointer to char	Standard
ammo_map_index	int	Standard
damage_file_type	unsigned char	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
ammo_fp	pointer to register FILE	
damage_file_name[80]	char	Standard
Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	procedure failed
TRUE	int	file opened successfully
Errors		
Error	**Reason for Error**	
stderr	- PANIC -- can't open file	
Calls		
Function	**Where Described**	
init_indirect_fire_table	Section 2.5.4.5.3	
init_direct_fire_table	Section 2.5.4.5.4	

Table 2.5-111: cfail_read_damage_file Information.

2.5.4.5.3 init_indirect_fire_table

This routine is used to initialize the indirect fire damage tables. First, it allocates memory for the table. Then, it reads the squared ranges, checking for a valid range of data. Finally, it calls **cfail_cdamages_init()** to read in the list of possible damages for each range and heading of indirect fire. The distance of the vehicle from the indirect fire explosion is categorized into one of four ranges based on the type of ammo: direct, near miss, far miss, or miss. No damage will occur outside of the miss range.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_fp	pointer to FILE	
ammo_type	int	Standard
ammo_file	pointer to char	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
range	REAL	/simnet/common/include/global/sim_types.h

Errors	
Error	**Reason for Error**
stderr	- FAIL: insufficient memory for combat tables - FAIL: unexpected eof found in file - FAIL: invalid range

Calls	
Function	**Where Described**
cfail_damages_init	Section 2.5.4.5.5

Table 2.5-112: init_indirect_fire_table Information.

2.5.4.5.4 init_direct_fire_table

This routine initializes the direct fire table.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_fp	pointer to FILE	
ammo_type	int	Standard
ammo_file	pointer to char	Standard

Errors	
Error	**Reason for Error**
stderr	- FAIL: insufficient memory for combat tables

Calls	
Function	**Where Described**
cfail_damages_init	Section 2.5.4.5.5

Table 2.5-113: init_direct_fire_table Information.

2.5.4.5.5 cfail_damages_init

This routine is used to initialize the table which contains normal combat damages. This routine reads in the bulk of the damages. In order to initialize the damage table the routine expects to be passed the index the the damage table, the index into the dispatch table, and the probability of a particular failure happening.

The damage table is set up under the following conditions: for any hit (consisting of a shell type, hit location, shell direction, and angle of incidence) there may be several possible failures. Each failure has an associated probability of occurrance and an index to a routine which is called if the failure occurs. For example, there may be a 30% chance that failure 4 occurs for a certain hit. If the 30% chance is true, then the routine indexed by number 4 is called. In this case, the radio antenna would be broken, and the tank's communciations would be affected accordingly.

To keep track of this information, an array of linked lists is created. The index to the array is formed by logically OR-ing together the various components of the hit (shell type, hit location, angle of incidence, etc.). The array contains a pointer to a list of possible failures for that hit. Each failure contains the routine index for the hit, the chance of occurrance, and a pointer to the next possible failure.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_fp	pointer to FILE	
damage_array[]	pointer to FAIL_TEST	Section 2.5.4.1.5
table_size	int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
fail_ptr	pointer to register FAIL_TEST	Section 2.5.4.1.5
index	int	Standard
ret	int	Standard
comment[COMMENT_SIZE]	char	Standard

Errors	
Error	**Reason for Error**
stderr	- FAIL: insufficient memory for combat tables

Table 2.5-114: cfail_cdamages_init Information.

2.5.4.6 cfail_loc.c
(./simnet/release/src/vehicle/libsrc/libfail/cfail_loc.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"fail.h"
"cfail.h"

Declaration:

damage_by_type[EFF_KIND_MASK]

2.5.4.7 cfail_loc.h
(./simnet/release/src/vehicle/libsrc/libfail/cfail_loc.h)

Defines:

INCIDENCE_0_30
INCIDENCE_30_60
INCIDENCE_60_90

HIT_FROM_LEFT
HIT_FROM_RIGHT
HIT_FROM_BACK
HIT_FROM_FRONT

HIT_ON_FRONT
HIT_ON_RIGHT
HIT_0N_BACK
HIT_ON_LEFT

HIT_ON_HULL
HIT_ON_TURRET

DIR_TABLE_SIZE

IND_SIDE_LEFT
IND_SIDE_REAR
IND_SIDE_RIGHT
IND_SIDE_FRONT

IND_RANGE_DIRECT
IND_RANGE_NEAR
IND_RANGE_FAR
IND_RANGE_MISS

IND_FUZE_PD
IND_FUZE_VT

IND_TABLE_SIZE

NUM_RANGES
RANGE_DIRECT
RANGE_NEAR
RANGE_FAR
RANGE_MISS

NO_TABLE
DIRECT_FIRE
INDIRECT_FIRE

Variable and Procedure Declarations:
cfail_cdamages_init()
cfail_check_damages()
cfail_get_composite_index()
damage_by_type[]
CFAIL_DEBUG

Typedefs:
DIR_AMMO_DAMAGES
IND_AMMO_DAMAGES
GEN_AMMO_DAMAGES

2.5.4.8 f_break_sys.c
(./simnet/release/src/vehicle/libsrc/libfail/f_break_sys.c)

Includes:
"stdio.h"
"math.h"
"pro_data.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libevent.h"
"libfail.h"
"libnetwork.h"
"librepair.h"
"fail.h"
"fail_loc.h"

2.5.4.8.1 fail_break_system

This routine is an external procedure which is called from outside libfail when it is determined that a system should break. It generates an *event_id* for the breakage, breaks the system, and notifies the network to send a StatusChangePDU.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
agent_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
cause	DamageCause	/simnet/common/include/protocol/p_data.h
system_num	int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
event_id	EventID	/simnet/common/include/protocol/basic.h
subsys	pointer to VehicleSubsystems	/simnet/common/include/protocol/status.h

Calls	
Function	**Where Described**
fail_system_is_broken	Section 2.5.4.8.2
fail_get_delta_subsystems	Section 2.5.4.14.7
event_get_eventid	Section 2.6.9.1.2
network_send_status_change	Section 2.1.1.3.1.62.1

Table 2.5-115: fail_break_system Information.

2.5.4.8.2 fail_system_is_broken

This routine is called when either the combat failures or stochastic failures parts of libfail determine that a system breaks. The parameter, *system_number*, signifies which system is to break.

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

Calls	
Function	**Where Described**
fail_set_subsys_bit	Section 2.5.4.14.8
repair_start_self_repair	Section 2.5.4.19.8

Table 2.5-116: fail_system_is_broken Information.

2.5.4.9 f_cat_kill.c
(./simnet/release/src/vehicle/libsrc/libfail/f_cat_kill.c)

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"pro_data.h"
"libevent.h"
"libfail.h"
"libsound.h"
"libnetwork.h"
"fail.h"
"fail_loc.h"

2.5.4.9.1 fail_cat_kill

This routine is called to break every subsystem when a catastrophic kill determination has been made externally to the libfail code.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
agent_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
cause	DamageCause	/simnet/common/include/protocol/p_data.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
event_id	EventID	/simnet/common/include/protocol/basic.h

Calls	
Function	**Where Described**
event_get_eventid	Section 2.6.9.1.2
fail_vehicle_is_destroyed	Section 2.5.4.9.2

Table 2.5-117: fail_cat_kill Information.

2.5.4.9.2 fail_vehicle_is_destroyed

This routine is called when a catastrophic kill determination has been made from a libfail routine.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
agent_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
event_id	EventID	/simnet/common/include/protocol/basic.h
cause	DamageCause	/simnet/common/include/protocol/p_sim.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
subsys	pointer to VehicleSubsystems	/simnet/common/include/protocol/p_data.h

Calls	
Function	**Where Described**
sound_we_just_died	Section 2.1.3.2.8
controls_electsys_dead	Sections 2.2.2, 2.3.2, and 2.4.2
controls_break_controls	Sections 2.2.2, 2.3.2, and 2.4.2
vision_break_all_blocks	Sections 2.2.6.4.3, 2.3.6.3.2, and 2.4.5.1
network_set_death_status	Section 2.1.1.3.1.11.1
network_set_smoking_status	Section 2.1.1.3.1.11.2
network_set_burning_status	Section 2.1.1.3.1.11.3
fail_set_sybsys_bit	Section 2.5.4.14.8
fail_get_delta_subsystems	Section 2.5.4.14.7
network_send_status_change	Section 2.1.1.3.1.62

Table 2.5-118: fail_vehicle_is_destroyed Information.

2.5.4.10 f_dth_stat.c
(./simnet/release/src/vehicle/libsrc/libfail/f_dth_stat.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"fail_loc.h"

2.5.4.10.1 fail_death_status

This routine tells whether you are dead or not.

Return Values		
Return Value	Type	Meaning
we_are_dead	int	we are dead

Table 2.5-119: fail_death_status Information.

2.5.4.11 f_init.c

(./simnet/release/src/vehicle/libsrc/libfail/f_init.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"address.h"
"basic.h"
"fail_loc.h"
"libfail.h"

2.5.4.11.1 fail_table_init

This routine is called to initialize the failure dispatch table. When a failure occurs, it is specified by a failure number as defined in /simnet/include/protocol/failure.h. The failure dispatch table maps the failure indices to the set of failure routines. This routine, however, initializes the dispatch table to all zeroes. Individual failures must be initialized through fail_init_failure(), below.

Internal Variables		
Variable	Type	Where Typedef Declared
i	int	Standard

Calls	
Function	Where Described
fail_subsys_init	Section 2.5.4.14.1

Table 2.5-120: fail_table_init Information.

2.5.4.11.2 fail_init_failure

This routine is called to initialize an entry in the failure dispatch table. The vehicle specific software calls this routine for each failure modeled. Each system must be initialized through this routine in order to be allowed to fail. The routine returns FALSE if any parameter information (*fail_num*, *self_repair*, or *summaryKill*) is out of bounds, and returns TRUE otherwise. Note that the failure and repair routine pointers can be NULL if desired, though this does not make the simulated failure very realistic. The *fail_num* parameter is the failure number as defined in the protocol file "failure.h". The *fail_rtn* and *repair_rtn* are pointers to the fail and repair routines. These routines actually cause the failure or repair to occur. *self_repair* is the self repair time (in minutes). Certain systems are summarized together, for example, the turret and gun are summarized as fire power systems. The *summaryKill* parameter tells which summary failures are mapped to the particular failure.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fail_num	int	Standard
fail_rtn	PFV	/simnet/common/include/global/sim_types.h
repair_rtn	PFV	/simnet/common/include/global/sim_types.h
self_repair	int	Standard
summaryKill	int	Standard

Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	either *self_repair* or *fail_num* parameter is invalid
TRUE	int	procedure was successful

Calls	
Function	**Where Described**
fail_failure_exists	Section 2.5.4.14.3

Table 2.5-121: fail_init_failure Information.

2.5.4.11.3 fail_init

This routine is called from libmain as the final initialization. It is used to reactivate a towed vehicle with its original failures intact. The initial failures are determined by the Vehicle Subsystems in the Activate packet and are broken by this routine.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
flag_num	int	Standard

Calls	
Function	**Where Described**
fail_reincarnation	Section 2.5.4.12.1
controls_restore_controls	Sections 2.2.2, 2.3.2, and 2.4.2
fail_is_component_broken	Section 2.5.4.14.6
fail_system_is_broken	Section 2.5.4.8.2

Table 2.5-122: fail_init Information.

2.5.4.12 f_reincarn.c

(./simnet/release/src/vehicle/libsrc/libfail/f_reincarn.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"fail_loc.h"

2.5.4.12.1 fail_reincarnation

This routine is a debugging tool used to revive a dead vehicle without reconstituting it.

Calls	
Function	**Where Described**
network_set_death_status	Section 2.1.1.3.1.11.1
network_set_smoking_status	Section 2.1.1.3.1.11.2
network_set_burning_status	Section 2.1.1.3.1.11.3

Table 2.5-123: fail_reincarnation Information.

2.5.4.13 f_simul.c
(./simnet/release/src/vehicle/libsrc/libfail/f_simul.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libevent.h"
"libnetwork.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"librva.h"
"pro_data.h"
"fail_loc.h"
"libfail.h"
"timers.h"

2.5.4.13.1 fail_simul

This routine checks to see if the vehicle is done burning or smoking and checks to see if the self-repair timers have timed out.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
burn_time	int	Standard
smoke_time	int	Standard

Calls	
Function	**Where Described**
network_set_smoking_status	Section 2.1.1.3.1.11.2
metwork_set_burning_status	Section 2.1.1.3.1.11.3
timers_set_null_timer	Section 2.6.3.14.1
timers_get_timeout_edge	Section 2.6.3.22.1
timers_free_timer	Section 2.6.3.5.1
fail_clear_subsys_bit	Section 2.5.4.14.9
network_send_status_change	Section 2.1.1.3.1.62.1

Table 2.5-124: fail_simul Information.

2.5.4.14 f_subsys.c

(./simnet/release/src/vehicle/libsrc/libfail/f_subsys.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"pro_data.h"
"status.h"
"libevent.h"
"fail.h"
"fail_loc.h"

This file contains routines to keep track of the protocol defined data structure, VehicleSubsystem, which tells which kill levels have been achieved and which components have been damaged. In most cases, the VehicleSubsystems are treated as arrays of ints in order to facilitate setting and clearing of important bits. In the VehicleSubsystems structure, space is allocated as follows:

<subsys1_exists>
<subsys1_status>
<subsys2_exists>
<subsys2_status>

Since failures are numbered sequentially without gaps, the gaps need to be included when calculating which bit to set (assuming that each subsystem is exactly 32 bits and that an int is 32 bits). Since the network protocol definition of a VehicleSubsystem is set up in a way that is difficult for libfail to directly set and clear bits while using the most convenient definitions for the simulation, the local type LocalSubsys is declared to be the same space as VehicleSubsystems, but is much simpler for libfail to access. The first element of each component array indicates if the component exists, and the second indicates component status. Both types use the subsystemExists and subsystemStatus as defined in "status.h".

This file defines numComponents.

This file contains a typedef of LocalSubsys.

This file declares:

per_subsys -- the permanent list of which subsystems are broken and which are not. If a component is broken it cannot be rebroken.

delta_subsys -- a local list used to keep a temporary running total of the broken subsystems. Used by failures and repairs to generate the "subsystems" field of the StatusChangePDU.

delta_changed -- a True/False variable, tells if delta_subsys has changed since last call to fail_get_delta_subsystems().

temp_subsys -- a local list used to keep a copy of delta_subsys for use by the outside world.

kill_levels[MaxNumFailures]
-- need to set to noKill (one per failure)

2.5.4.14.1 fail_subsys_init

This routine initializes the fail systems, checking that an unsigned long integer is 32 bits.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard

Table 2.5-125: fail_subsys_init Information.

2.5.4.14.2 fail_set_subsys

This routine sets the initial permanent subsystems by copying the failures array into the local space. The failures which correspond to the broken components are then initialized from **fail_init()**, in "f_init.c".

Parameters		
Parameter	**Type**	**Where Typedef Declared**
new_subsys	pointer to VehicleSubsystems	/simnet/common/include/protocol/p_data.h

Table 2.5-126: fail_set_subsys Information.

2.5.4.14.3 fail_failure_exists

This routine is called from **fail_init_failure()** to indicate that a particular failure should be enabled. The subsystemExists bit corresponding to the failure is set to indicate that the subsystem exists and is capable of that failure. The *summary_kill* parameter indicates to which summary failures (if any) this particular failure is mapped. The *summary_kill* levels associated with the failure are drawn from "status.h" and defined in "libfail.h".

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fail_num	int	Standard
summary_kill	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
subsys_num	int	Standard
subsys_bit	int	Standard

Table 2.5-127: fail_failure_exists Information.

2.5.4.14.4 fail_clear_subsys

This routine clears out a subsystems summary and component status list. It is assumed that the subsystemExists array will be set only once.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
subsys	pointer to LocalSubsys	
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard

Table 2.5-128: fail_clear_subsys Information.

2.5.4.14.5 fail_get_perm_subsys

This routine is called by the network to get a complete list of what is broken and the specific failures due to combat action. The routine gives a pointer to the failure flags so the network can send a list of failures in the VehicleStatusPDU.

Return Values		
Return Value	**Type**	**Meaning**
&perm_subsys	pointer to VehicleSubsystems	/simnet/common/include/protocol/p_data.h

Table 2.5-129: fail_get_perm_subsys Information.

2.5.4.14.6 fail_is_component_broken

This routine is called to determine if a particular *fail_num* is broken. The routine returns FALSE if the failure does not exist or is not broken, and returns TRUE if the failure both exists and is broken.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fail_num	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
subsys_num	int	Standard
subsys_bit	usigned long int	Standard
Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	either failure does not exist or is not broken
TRUE	int	failure exists and is broken

Table 2.5-130: fail_is_component_broken Information.

2.5.4.14.7 fail_get_delta_subsystems

This routine is called when a statusChangePDU is sent. It returns a pointer to the VehicleSubsystems that have changed since the last call to this routine. If nothing has changed since then, it returns NULL.

Return Values		
Return Value	**Type**	**Meaning**
NULL	pointer to VehicleSubsystems	no changes to VehicleSubsystems
&temp_subsys	pointer to VehicleSubsystems	the changes to VehicleSubsystems since the last call

Calls	
Function	**Where Described**
fail_clear_subsys	Section 2.5.4.14.4

Table 2.5-131: fail_get_delta_subsystems Information.

2.5.4.14.8 fail_set_subsys_bit

This routine is called within libfail to set the failure flags associated with a particular subsystem. The failure flag is specified by the parameter *fail_num*. The routine returns FALSE if *fail_num* is invalid or if the system was already broken, and returns TRUE otherwise. particular subsystem specified. sets particular flags bit. checks for summaryKill bit.

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

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
subsys_num	int	Standard
subsys_bit	unsigned long int	Standard

Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	invalid *fail_num* or system is already broken
TRUE	int	procedure was sucessful

Table 2.5-132: fail_set_subsys_bit Information.

2.5.4.14.9 fail_clear_subsys_bit

This routine is called by repairs to reset the failure flag associated with a particular subsystem repair in order to indicate that the subsystem has been repaired. The particular failure to clear is specifed by the parameter *fail_num*. Returns FALSE if *fail_num* is invalid or if the component was not broken; returns TRUE otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fail_num	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
subsys_num	int	Standard
subsys_bit	unsigned long int	Standard
Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	*fail_num* is invalid or the component was not broken
TRUE	int	procedure was successful

Table 2.5-133: fail_clear_subsys_bit Information.

2.5.4.15 fail.h
(./simnet/release/src/vehicle/libsrc/libfail/fail.h)

Typedef:

FAIL_TEST -- the structure's component consists of:
rtn_index - the routine used to implement the failure
prob - the probability of failure
next - the next failure to occur

2.5.4.16 fail_loc.c
(./simnet/release/src/vehicle/libsrc/libfail/fail_loc.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"fail_loc.h"
"libfail.h"

Declarations:

we_are_dead -- tells if we are dead
repair_timers[NUM_SELF_REPAIR_TIMERS]
-- instanciation of a self repair timer structure.
dispatch_table[MaxNumFailures]
SELF_REPAIR_TIMER
-- an externally declared timer which is started when the self repair starts to count down. This timer tells how long the repair takes and which system gets repaired.
FAIL_INFO -- Every system able to fail has one of these structures, where:
fail_rtn tells which routine implements the failure
repair_rtn tells which routine to fix the failure,
self_repair_timer gives a countdown time if applicable

2.5.4.17 fail_loc.h
(./simnet/release/src/vehicle/libsrc/libfail/fail_loc.h)

Include "libfail.h"

Defines:

NUM_SELF_REPAIR_TIMERS
MaxNumFailures
BURNING_TIME -- burn for 15 minutes after being killed
SMOKING_TIME -- smoke for 30 minutes after being killed

Typedefs:

SELF_REPAIR_TIMER
FAIL_INFO

Declarations:

repair_timers[]
we_are_dead

dispatch_table[]
fail_failure_exists()
fail_is_component_broken()
fail_set_subsys_bit()
fail_clear_subsys_bit()
fail_init_flags()
fail_cat_kill()
fail_break_system()
fail_break_flag()
fail_repair_flag()
fail_start_self_repair()

2.5.4.18 rand.c

(./simnet/release/src/vehicle/libsrc/libfail/rand.c)

This file contains two different standard algorithm choices for generating random numbers.

2.5.4.19 repair.c

(./simnet/release/src/vehicle/libsrc/libfail/repair.c)

This file contains the repairs functionality. Repairs can be classified in the following two ways:

1) The MCC Maintenance console may arrange with the crew to send a repair truck to the vehicle to perform the repair.
2) Self-repairs in which the crew can repair certain failures themselves (usually deterministic failures). The repairs are timed through the self-repair timer; the failure fixes itself after the set amount of time.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"librva.h"
"libnetwork.h"
"llibrepair.h"
"libevent.h"
"fail_loc.h"
"timers.h"

Declarations:

repair_mapping -- The set of mapping between failures and repairs. One repair may fix more than one failure. The mapping is set up in the vehicle specific code.

num_replace_repair

2.5.4.19.1 lrepair_init

This routine is called by the vehicle specific code to insert the set of mapping.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_dependent_mapping	pointer to short	Standard
num_maps	int	Standard

Table 2.5-134: lrepair_init Information.

2.5.4.19.2 repair_uninit

This routine is called to uninitialize a vehicle in order to clear out the self-repair timers.

Internal Variables		
Variable	Type	Where Typedef Declared
i	int	Standard
Calls		
Function	Where Described	
timers_set_sub_timer	Section 2.6.3.14.1	
tiemrs_free_timer	Section 2.6.3.5.1	

Table 2.5-135: repair_uninit Information.

2.5.4.19.3 repair_fix_system

This routine is called from external to libfail in order to repair a system.

Parameters		
Parameter	Type	Where Typedef Declared
cause	DamageCause	/simnet/common/include/protocol/p_data.h
repair_code	int	Standard
Return Values		
Return Value	Type	Meaning
repair_system_is_fixed (&vehicleIDconstant, event_get_eventid(NO_SKIP), cause, repair_code)	int	whether valid repair code: if TRUE then repair code is valid if FALSE then repair code is invalid
Calls		
Function	Where Described	
repair_system_is_fixed	Section 2.5.4.19.4	
event_get_eventid	Section 2.6.9.1.2	

Table 2.5-136: repair_fix_system Information.

2.5.4.19.4 repair_system_is_fixed

This routine is called internally from libfail in order to repair a system.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
agent_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
event_id	EventID	/simnet/common/include/protocol/basic.h
cause	DamageCause	/simnet/common/include/protocol/p_data.h
repair_code	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
fail_sys	int	Standard
subsys	pointer to VehicleSubsystems	/simnet/common/include/protocol/p_data.h
Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	*repair_code* is invalid
TRUE	int	*repair_code* is valid

Calls	
Function	**Where Described**
fail_clear_subsys_bit	Section 2.5.4.14.9
fail_get_delta_subsystems	Section 2.5.4.14.7
network_send_status_change	Section 2.1.1.3.1.62.1

Table 2.5-137: repair_system_is_fixed Information.

2.5.4.19.5 repair_fix_failure

This routine allows the repair system to be bypassed in order to fix a specific failure. The routine is used when the failures to repairs mapping is not needed.

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

Calls	
Function	**Where Described**
fail_clear_subsys_bit	Section 2.5.4.14.9

Table 2.5-138: repair_fix_failure Information.

2.5.4.19.6 repair_complete_system

This routine fixes the complete system when the self-repair timers reach zero.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
sys_num	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
subsys	pointer to VehicleSubsystems	/simnet/common/include/protocol/p_data.h

Calls	
Function	**Where Described**
timers_free_timer	Section 2.6.3.5.1
timers_set_null_timer	Section 2.6.3.14.1
fail_clear_subsys_bit	Section 2.5.4.14.9
fail_get_delta_subsystems	Section 2.5.4.14.7
network_send_status_change	Section 2.1.1.3.1.62.1
event_get_eventid	Section 2.6.9.1.2

Table 2.5-139: repair_complete_system Information.

2.5.4.19.7 repair_all_systems

This routine is called when the vehicle is reincarnated in order to bypass the repair facility and call all repair routines. The routine may only be called from the keyboard.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
subsys	pointer to VehicleSubsystems	/simnet/common/include/protocol/p_data.h

Calls	
Function	**Where Described**
fail_reincarnation	Section 2.5.4.12.1
controls_restore_controls	Sections 2.2.2, 2.3.2, and 2.4.2
vision_restore_all_blocks	Sections 2.2.6.4.3, 2.3.6.3.2, and 2.4.5.1
fail_clear_subsys_bit	Section 2.5.4.14.9
fail_get_delta_subsystems	Section 2.5.4.14.7
network_send_status_change	Section 2.1.1.3.1.62.1
event_get_eventid	Section 2.6.9.1.2

Table 2.5-140: repair_all_systems Information.

2.5.4.19.8 repair_start_self_repair

This routine allocates and starts the self-repair timers when a system with self-repair timers has a failure.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
system_number	int	Standard
time_to_repair	int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
timer_to_use	int	Standard

Calls	
Function	**Where Described**
timers_free_timer	Section 2.6.3.5.1
timers_set_null_timer	Section 2.6.3.14.1
timers_get_timer	Section 2.6.3.6.1

Table 2.5-141: repair_start_self_repair Information.

2.5.4.20 s_curr_cond.c
(./simnet/release/src/vehicle/libsrc/libfail/s_curr_cond.c)

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"sfail_loc.h"

2.5.4.20.1 get_curr_condition

Stochastic failures depend upon the condition of the vehicle. The condition is determined by the mean miles between failures (MMBF) and the maintenance level of the vehicle. If the MMBF is lowered, the probability that damage will occur increases. A maintenance level is assigned to each vehicle based on the age of the vehicle (i.e. a brand new tank is assigned the lowest maintenance level of 1 and a 5+ years old tank is assigned the highest maintenance level of 5). Note that a maintenance level of 1 is assumed when initializing.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
best_mmbf	int	Standard
maint_level	int	Standard

Return Values		
Return Value	**Type**	**Meaning**
best_mmbf main_level_ratios[maint_level]	int	the current condition of vehicle

Errors	
Error	**Reason for Error**
stderr	FAIL: get_curr_condition: invalid maint_level

Table 2.5-142: get_curr_condition Information.

2.5.4.21 s_event.c

(./simnet/release/src/vehicle/libsrc/libfail/s_event.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"status.h"
"fail_loc.h"
"sfail_loc.h"
"libnetwork.h"
"libevent.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"librva.h"

2.5.4.21.1 sfail_event_occurred

This routine gets called when the stochastic failures event has occurred. This event is usually either that the vehicle has traveled a certain distance or a certain number of rounds of gunfire have been fired by the vehicle. This routine checks to see that the event has occurred, rolls the dice, and checks the failure table for the appropriate failure. A message is sent on the network indicating any failures.

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

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
system_num	register int	Standard
percent	register int	Standard
sub_sub	pointer to register FAIL_TEST	Section 2.5.4.15
subsys	pointer to VehicleSubsystems	/simnet/common/include/protocol/p_data.h

Errors	
Error	**Reason for Error**
stderr	- sfail in subsystem: # - sfail: system is broken

Calls	
Function	**Where Described**
roll_dice	Macro defined in /simnet/common/include/global/sim_macros.h
rand	Section 2.5.4.18
fail_system_is_broken	Section 2.5.4.8.2
fail_get_delta_subsystems	Section 2.5.4.14.7
network_send_status_change	Section 2.1.1.3.1.62.1

Table 2.5-143: sfail_event_occurred Information.

2.5.4.22 s_fixed.c
(./simnet/release/src/vehicle/libsrc/libfail/s_fixed.c)

Include:

"stdio.h:
"math.h"
"sim_types.h"
"sim_dfns.h:
"sim_macros.h"
"sfail_loc.h"

2.5.4.22.1 sfail_fixed_good_as_new

This routine sets the current condition for a replaced subsystem to a maintenance level of 1.

Parameters		
Parameter	Type	Where Typedef Declared
subsystem	int	Standard

Calls	
Function	Where Described
get_curr_condition	Section 2.5.4.20.1

Table 2.5-144: sfail_fixed_good_as_new Information.

2.5.4.23 s_init.c
(./simnet/release/src/vehicle/libsrc/libfail/s_init.c)

Include:

"stdio.h"
"math.h" (Simnet Butterfly Machine only)
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"simstdio.h"
"libfail.h"
"fail.h"
"sfail_loc.h"

Defines:

COMMENT_SIZE

2.5.4.23.1 sfail_init

This routine sets up and initializes the stochastic failures damage table. A vehicle is assumed to brand new with a maintenance level of 1 when initialized.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
sdam_file	pointer to char	Standard
num_sub_sys	int	Standard
num_maint_levels	int	Standard
veh_maint_levels[]	REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
fp	pointer to FILE	
i	int	Standard
system_num	int	Standard
num_sfails	int	Standard
curr_mmbf	int	Standard
event	int	Standard
sub_sub_sys	pointer to FAIL_TEST	Section 2.5.4.15
comment[COMMENT_SIZE]	char	Standard

Errors	
Error	**Reason for Error**
stderr	- PANIC -- can't open damage file - FAIL -- insufficient memory for sfail tables - FAIL -- unexpected eof in file

Calls	
Function	**Where Described**
get_curr_condition	Section 2.5.4.20.1

Table 2.5-145: sfail_init Information.

2.5.4.24 sfail_loc.c
(/simnet/release/src/vehicle/libsrc/libfail/sfail_loc.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"sfail_loc.h"
"fail.h"

Declarations:

maint_level_ratios[MAX_NUM_MAINT_LEVELS]
-- sets the maintenance levels into ratios. The maintenance levels are all relative to maint_level_ratios[1], which is equal to 1.0. The other maintenance levels have ratio equivelants, such that if maint_level_ratios[n] = 0.5, the vehicle is twice as likely to suffer a stochastic failure. maint_level_ratios[n] corresponds to maintenance level n, therefore maint_level_ratios[0] = 0.0, since no maintenance level 0 exists.
sub_system[MAX_NUM_SUBSYSTEMS]
NUM_MAINT_LEVELS
NUM_SUB_SYSTEMS

Defines:

MAX_NUIM_SUBSYSTEMS

2.5.4.25 sfail_loc.h
(/simnet/release/src/vehicle/libsrc/libfail/sfail_loc.h)

Includes:

"fail.h"

Defines:

SFAIL_DEBUG
MAX_NUM_MAINT_LEVELS

Typedefs:

sfail_type -- the structure of the stochastic failure array entry, where: *mmbf* is the mean miles between failures when new, *curr_mmbf* is the current mean miles between failures, *event* is the event that causes a stochastic failures check, *fail_list* is the list of failures that may occur

External declarations:

sub_system[]
NUM_MAINT_LEVELS
NUM_SUB_SYSTEMS
maint_level_ratios[]

2.5.4.26 sfail_mnt_cond.c
(./simnet/release/src/vehicle/libsrc/libfail/sfail_mnt_cond.c)

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"sfail_loc.h"

The variable init_maint_condition is declared.

2.5.4.26.1 sfail_maint_cond

This routine returns the current maintenance condition of the vehicle.

Return Values		
Return Value	**Type**	**Meaning**
1	int	the initial maintenance level is set at 1, signifying a brand new vehicle
init_maint_condition	int	the maintenance level
Errors		
Error	**Reason for Error**	
stderr	PANIC -- in sfail -- bad initial maint level	

Table 2.5-146: sfail_maint_cond Information.

2.5.4.26.2 sfail_maintenance_condition

This routine is called to change the mainetance level for the vehicle. In this routine, the current level is set to the level passed in the parameter, *condition*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
condition	register int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
sytem_num	register int	Standard
Errors		
Error	**Reason for Error**	
stderr	PANIC -- in sfail -- bad maint level	
Calls		
Function	**Where Described**	
get_curr_condition	Section 2.5.4.20.1	

Table 2.5-147: sfail_maintenance condition Information.

2.5.5 libturret

(./simnet/release/src/vehicle/libsrc/libturret [libturret])

More fidelity is required of the model of the turret and gun than is required for the other moving components. The basic model for the M1 and M2 turrets is found in libturret. Functions are provided to traverse the turret and elevate the gun. These functions perform stabilization of the gun as well. Messages are put on the network by libturret to inform the world of turret position.

2.5.5.1 libturret.h

(./simnet/release/src/vehicle/libsrc/libturret/libturret.h)

This file contains the Turret Simulation Module and includes the stabilization system and gunner's primary sight.

Includes:
"basic.h"

External Declarations:
gps_in_world
hull_to_turret
turret_to_hull
turret_to_sight
sight_to_turret
sight_to_world
gun_to_turret
turret_to_gun

set_turret_vars()
turret_elevate_gun()
turret_elevate_sight()
turret_get_azimuth_str()
turret_get_gun_tip()
turret_get_gun_to_world()
turret_get_ref_ind()
turret_get_network_azimuth()
turret_get_network_elevation()
turret_calc_azimuth()
turret_get_total_turret_slew_rate()
turret_get_total_gun_elev_rate()
turret_get_sight_in_world()
turret_get_stab_changes()
turret_move_azimuth()
turret_move_elevation()
turret_null_azimuth_ind
turret_pos_init()
turret_send_azimuth_ind()
turret_set_stab_sys()
turret_set_stab_vector()
turret_stops_init()
turret_sync_hun_with_sight
turret_sybnc_sight_with_gun
turret_update_check()
turret_update_rva()

2.5.5.2 turret.c

(./simnet/release/src/vehicle/libsrc/libturret/turret.c)

Contains the simulation of generic turret functions. This file maintains the matrices which contain the orientations of the turret and gun .

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"basic.h"
"libmatrix.h"
"libkin.h"
"libhull.h"

Defines:

TURRET_DEBUG
TURRET_FAILURES_DEBUG
STAB_DEBUG

ON_TOP_STOP	-- options for sight_on_stop and gun_on_stop
ON_BOTTOM_STOP	
NOT_STOPPED	
TURRET_AZI_DEGS	-- in degrees
TURRET_AZI_CHANGE	
GUN_ELEV_DEGS	-- in degrees
GUN_ELEV_CHANGE	

Declarations:

hull_to_turret
turret_to_hull
sight_to_turret
sight_to_world
turret_to_sight
gun_to_turret
turret_to_gun
SIGHT_MAX_ELEV
SIGN_MIN_ELEV
GUN_MAX_ELEV
GUN_MIN_ELEV
sight_on_stop
gun_on_stop

sight_in_hull	-- normalized vector used by the stabilization system (sight in hull coordinates)
sight_in_world	-- normalized vector used by the stabilization system (sight in world coordinates)
turret_azimuth	-- the number of radians that the turret is rotated in azimuth relative to the hull. An increasing angle means a clockwise rotation. 0.0 means that the turret is aligned with the hull. 0.0 <= turret_azimuth < 2*PI

total_turret_slew_rate -- in radians per tick
total_gun_elev_rate -- in radians per tick
gun_elevation -- the number of radians that the gun is elevated relative to the hull. An increasing angle means an elevation. 0.0 means that the gun is parallel to the tank hull.
0.0 <= gun_elevation < 2*PI
time_to_update_rva
delta_elevation
delta_azimuth
azimuth_str[80]
azimuth_range_format
send_grid_azimuth
w_to_g_mat
super_elev_mat
lead_track_mat

turret_get_slope_ind()
turret_get_ref_ind()
turret_calc_azimuth()
turret_get_stab_changes()
turret_move_azimuth()
elevate_system()
set_turret_vars()

2.5.5.2.1 turret_stops_init

This routine initializes the locations of the gun and sight stops. All parameters represent the sines of the angles.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
sight_max	REAL	/simnet/common/include/global/sim_types.h
sight_min	REAL	/simnet/common/include/global/sim_types.h
gun_max	REAL	/simnet/common/include/global/sim_types.h
gun_min	REAL	/simnet/common/include/global/sim_types.h

Table 2.5-148: turret_stops_init Information.

2.5.5.2.2 turret_pos_init

This routine initializes the turret azimuth variables, initializes the stabilization vectors so that the stabilization system can be used as soon as needed, and creates the identity matrices. The parameter, *init_turret_azimuth*, represents the initial azimuth of the turret relative to the hull in a circle ranging in value from 0 to 0xffffffff, where 0 is aligned perfectly with the front of the hull, and the values increase as the turret is initialized in a counterclockwise direction from the heading of the hull. The variable, *rad_azimuth*, represents the azimuth in radians, clockwise from the hull.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
init_turret_azimuth	usigned int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
rad_azimuth	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_rot_init	Section 2.6.2.47.1
mat_transpose	Section 2.6.2.51.1
mat_ident_init	Section 2.6.2.31.1

Table 2.5-149: turret_pos_init Information.

2.5.5.2.3 turret_set_stab_sys

This routine is called after the turret simulation is completed. After the turret has moved, the vector it will point to is saved to be set in the next tick. It forms two vectors which represent the sight vector in both hull and world coordinates. The sight to world matrix is generated in the process. This is called by **terrain_get_stab_elev**() and **terrain_get_stab_rot**().

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
h_to_w	T_MATRIX	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
vec_init	Section 2.6.2.61.1
mat_mat_mul	Section 2.6.2.32.1
vec_mat_mul	Section 2.6.2.56.1
mat_copy	Section 2.6.2.39.1
kinematics_get_h_to_w	Section 2.5.8.2.2

Table 2.5-150: turret_set_stab_sys Information.

2.5.5.2.4 turret_set_stab_vector

This routine is not used in the version 6.6 release.

2.5.5.2.5 turret_get_stab_changes

After the hull is moved, and before the turret is moved, this routine is called by **turret_move()** to determine how far the turret should be rotated and how high the gun should be elevated, in order to compensate for hull movement in the current frame. The parameter, *azimuth_rot*, represents the sine in radians of the rotation. The variables *old_sight* and *new_sight* represent the angles the turret must move to in order to align the sight to the last position.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
azimuth_rot	pointer to REAL	/simnet/common/include/global/sim_types.h
elev_rot	pointer to REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
old_sight	register pointer REAL	/simnet/common/include/global/sim_types.h
new_sight	register pointer REAL	/simnet/common/include/global/sim_types.h
old_dot_new	register REAL	/simnet/common/include/global/sim_types.h
sqr_norm_A_norm_B	register REAL	/simnet/common/include/global/sim_types.h
sqr_cos_rotation	register REAL	/simnet/common/include/global/sim_types.h
sin_rotation	register REAL	/simnet/common/include/global/sim_types.h
x_prod	register REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
vec_mat_mul	Section 2.6.2.56.1
kinematics_get_w_to_h	Section 2.5.8.2.2

Table 2.5-151: turret_get_stab_changes Information.

2.5.5.2.6 turret_move_azimuth

This routine moves the turret in azimuth given the total slew rate. The change over the current tick with respect to the world is added to the azimuth. The RVA table is updated. The turret to hull matrix is updated and the new azimuth is calculated.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
total_slew_rate	REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
rot_matrix	T_MATRIX	/simnet/common/include/global/sim_types.h
t_to_h	register T_MAT_PTR	/simnet/common/include/global/sim_types.h
h_to_t	register T_MAT_PTR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_rot_init2	Section 2.6.2.33.1
mat_mat_mul	Section 2.6.2.32.1
mat_transpose	Section 2.6.2.51.1

Table 2.5-152: turret_move_azimuth Information.

2.5.5.2.7 turret_move_elevation

This routine moves the gun in elevation, given the total elevation rate. In addition to elevating the gun, this routine is also responsible for checking to make sure that the gun is not moving too fast. It also checks to see that sufficient hydraulic pressure is available before actually elevating the gun. If the gun is slaved to sight, the gun is aligned with the sight. x = sin(x) approximation is used in the rotation matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
total_elev_rate	register REAL	/simnet/common/include/global/sim_types.h
gun_slaved_to_sight	int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
sight_elev_mat	T_MATRIX	/simnet/common/include/global/sim_types.h
s_e_mat	register T_MAT_PTR	/simnet/common/include/global/sim_types.h
t_to_sight	register T_MAT_PTR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_rot_init2	Section2.6.2.17.1
mat_mat_mul	*Section* 2.6.2.32.1
mat_transpose	Section 2.6.2.51.1
mat_copy	Section 2.6.2.39.1

Table 2.5-153: turret_move_elevation Information.

2.5.5.2.8 turret_elevate_sight

This routine returns TRUE if the elevation was successful, and FALSE if the sight hits either the top or bottom stop.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
elev_rate	register REAL	/simnet/common/include/global/sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
sight_on_stop == NOT_STOPPED	int	if TRUE, the elevation was successful; if FALSE, the sight hit either the top or bottom stop
Calls		
Function	**Where Described**	
elevate_system	Section 2.5.5.2.10	

Table 2.5-154: turret_elevate_sight Information.

2.5.5.2.9 turret_elevate_gun

This routine returns TRUE if the elevation was successful, and FALSE if the gun hits either the top or bottom stop.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
elev_rate	register REAL	/simnet/common/include/global/sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
gun_on_stop == NOT_STOPPED	int	if TRUE, the elevation was successful; if FALSE, the gun hit either the top or bottom stop
Calls		
Function	**Where Described**	
elevate_system	Section 2.5.5.2.10	

Table 2.5-155: turret_elevate_gun Information.

2.5.5.2.10 elevate_system

This routine is called by the routine **elevate_sight()** to calculate the sight elevation and by the routine **elevate_gun()** to calculate the gun elevation. The system (either the gun or the sight) is moved to the desired elevation. The routine checks to see if the system has hit one of the stops. If the desired elevation is beyond a stop, the system is moved to the stop.

and to check for the system hitting the stops.

elev_amount -- the sine of the desired elevation angle
top_stop -- the sine of the top stop angle
bottom_stop -- the sine of the bottom stop angle
stop_statua -- whether the system is at one of the stops

Parameters		
Parameter	**Type**	**Where Typedef Declared**
turret_to_system	register T_MATRIX	/simnet/common/include/global/sim_types.h
system_to_turret	register T_MATRIX	/simnet/common/include/global/sim_types.h
elev_amount	register REAL	/simnet/common/include/global/sim_types.h
top_stop	REAL	/simnet/common/include/global/sim_types.h
bottom_stop	REAL	/simnet/common/include/global/sim_types.h
stop_status	pointer to int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
system_elev_mat	T_MATRIX	/simnet/common/include/global/sim_types.h
s_e_mat	register T_MAT_PTR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_rot_init2	Section 2.6.2.33.1
mat_mat_mul	Section 2.6.2.32.1
mat_transpose	Section 2.6.2.51.1

Table 2.5-156: elevate_system Information.

2.5.5.2.11 turret_sync_gun_with_sight

This routine is called by the vehicle specific turret code to move the gun and the sight back together after they have been apart, given the offset of the gun from the sight. *difference* is the sine of the angle between the gun and the sight. An example of when this routine is called is after the gun has swung low over the back deck, using back deck clearance.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
difference	REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
difference_matrix	T_MATRIX	/simnet/common/include/global/sim_types.h
d_mat	T_MAT_PTR	/simnet/common/include/global/sim_types.h
Calls		
Function	**Where Described**	
mat_rot_init2	Section 2.6.2.33.1	
mat_mat_mul	Section 2.6.2.32.1	
mat_transpose	Section 2.6.2.51.1	

Table 2.5-157: turret_synch_gun_with_sight Information.

2.5.5.2.12 turret_synch_sight_with_gun

This routine is called by the vehicle specific turret code to move the sight and the gun back together after they have been apart, given the offset of the sight from the gun. For example, after the gun has hit its top stop, the sight may still be able to move up, even though it should not. This routine will be used to correct the instantaneous problem that results from the gun hitting its stop. *difference* is the sine of the angle between the gun and the sight.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
difference	REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
difference_matrix	T_MATRIX	/simnet/common/include/global/sim_types.h
d_mat	T_MAT_PTR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_rot_init2	Section 2.6.2.33.1
mat_mat_mul	Section 2.6.2.32.1
mat_transpose	Section 2.6.2.51.1

Table 2.5-158: turret_sync_sight_with_gun Information.

2.5.5.2.13 turret_get_g_to_w

This routine is called by gunnery when the gun is about to be fired. It determines the gun to world matrix, taking into account the lead azimuth and super elevation of the gun at the moment of firing (these variables are obtained by calls to the ballistics computer).

First, the turret_to_world matrix is found by multiplying turret_to_hull by hull_to_world. This matrix is rotated in the plane of the turret to compensate for the lead azimuth. Then, the gun is elevated, accounting for both the sight elevation and the super elevation from the ballistics computer. The rotations must be calculated in this order, rather than starting with the gps vector, rotating for lead azimuth, and elevating for superelevation since the rotation for lead azimuth is done around the gps Z axis, not the turret Z axis. Since problems would occur when the gun tube was highly elevated, this set of rotations is not being used. Note: the ballistics system is not meant to adjust for the tank pitched.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
g_to_w	register T_MATRIX	/simnet/common/include/global/sim_types.h
lead_azimuth	register REAL	/simnet/common/include/global/sim_types.h
super_elevation	register REAL	/simnet/common/include/global/sim_types.h
error_offset	register pointer REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
w_to_g	register T_MAT_PTR	/simnet/common/include/global/sim_types.h
super_elev	register T_MAT_PTR	/simnet/common/include/global/sim_types.h
lead_track	register T_MAT_PTR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_mat_mul	Section 2.6.2.32.1
kinematics_get_w_to_h	Section 2.5.8.2.1
mat_rot_init2	Section 2.6.2.33.1
mat_transpose	Section 2.6.2.51.1

Table 2.5-159: turret_get_g_to_w Information.

2.5.5.2.14 turret_get_network_elevation

This routine is called by the network to get the gun elevation. It returns an unsigned long which represents the gun elevation, where 0 means the gun is parallel with the hull of the tank, increasing as the gun elevates.

Because of constraints on precision in the Butterfly machine, this routine computes the elevation as half the angle (by dividing the unsigned long value by 4PI rather than 2PI), then multiplying by 2 by left shifting by 1.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
long_elev	Angle	/simnet/common/include/global/basic.h
Return Values		
Return Value	**Type**	**Meaning**
long_elev	Angle	the gun elevation

Table 2.5-160: turret_get_network_elevation Information.

2.5.5.2.15 turret_get_network_azimuth

This routine is called by the network to get the turret_azimuth. It returns an unsigned long which represents the turret azimuth, where 0 means that the turret is aligned with the front of the tank, increasing as the turret rotates counterclockwise as viewed looking down on the tank.

Because of constraints on precision in the Butterfly machine, this routine computes the azimuth as half the angle (by dividing the unsigned long value by 4PI rather than 2PI), then multiplying by 2 by left shifting by 1.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
long_azi	Angle	/simnet/common/include/global/basic.h
Return Values		
Return Value	**Type**	**Meaning**
long_azi	Angle	the turret azimuththe turret azimuth

Table 2.5-161: turret_get_network_azimuth Information.

2.5.5.2.16 turret_get_ref_ind

This routine is called to tell controls the azimuth of the turret relative to the hull in radians.

Return Values		
Return Value	**Type**	**Meaning**
turret_azimuth	REAL	The turret azimuth in radians

Table 2.5-162: turret_get_ref_ind Information.

2.5.5.2.17 turret_null_azimuth_ind

This routine sets *send_grid_azimuth* to FALSE.

2.5.5.2.18 turret_send_azimuth_ind

This routine se's *send_grid_azimuth* to TRUE.

2.5.5.2.19 turret_get_azimuth_str

This routine determines whether to notify controls of the turret azimuth relative to the world. Controls should be notified when the vehicle is stopped and the commander has pushed the appropriate button. The azimuth is calculated and converted to a character string.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
azimuth	REAL	/simnet/common/include/global/sim_types.h
azi_str	register pointer char	Standard
Return Values		
Return Value	**Type**	**Meaning**
azi_str	pointer to char	The turret azimuth as a character string
Calls		
Function	**Where Described**	
turret_calc_azimuth		

Table 2.5-163: turret_get_azimuth_str Information.

2.5.5.2.20 turret_update_check

If *time_to_update_rva* is non-zero, the RVA is updated.

Return Values		
Return Value	**Type**	**Meaning**
time_to_update_rva	int	Standard

Table 2.5-164: turret_update_check Information.

2.5.5.2.21 turret_update_rva

This routine sets the elevation and azimuth changes to 0.0, and sets *time_to_update_rva* equal to FALSE.

2.5.5.2.22 turret_get_sight_in_world

This routine returns the sight_in_world.

Return Values		
Return Value	**Type**	**Meaning**
sight_in_world	pointer to REAL	/simnet/common/include/global/sim_types.h

Table 2.5-165: turret_get_sight_in_world Information.

2.5.6 libsusp
(/simnet/release/src/vehicle/libsrc/libsusp [libsusp])

The suspensions of the M1 and M2 simulators are simulated by assuming a suspension consisting of one linear spring-damper assembly for each track, and one rotational spring-damper assembly between the hull and the undercarriage. This model accommodates one linear degree of freedom in the Z direction (up/down), and two rotational degrees of freedom, pitch and roll. This functionality is realized by one CSU, libsusp.

This library provides services for setting the damped natural frequency, and damping ratio for each of the three assemblies, setting the acceleration of the chassis, and setting a gun force reaction. These services are used by libbigwheel which maintains the relationship between the vehicle and the terrain.

2.5.6.1 gun_fired.c
(/simnet/release/src/vehicle/libsrc/libsusp/gun_fired.c)

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"susp_loc.h"

2.5.6.1.1 suspension_gun_fired

This routine is called to model the suspension when the gun is fired. *out_susp* is the structure to be operated on; *t_cos* is the cosine of the gun angle with respect to the hull; *t_sin* is the sine of the gun angle with respect to the hull. The *gun_fired* flag is set to TRUE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_susp	pointer to int	Standard
t_cos	REAL	/simnet/common/include/global/sim_types.h
t_sin	REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
loc_susp	pointer to SUSPENSION	susp_loc.h

Table 2.5-166: suspension_gun_fired Information.

2.5.6.2 libsusp.h
(/simnet/release/src/vehicle/libsrc/libsusp/libsusp.h)

External function declarations:

suspension_gun_fired()
suspension()
suspension_params()
suspension_init()
suspension_uninit()
suspension_veh_init()
suspension_acceleration_is()

2.5.6.3 susp_accel.c
(/simnet/release/src/vehicle/libsrc/libsusp/susp_accel.c)

Includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"susp_loc.h"

2.5.6.3.1 suspension_acceleration_is

This routine passes in the vehicle acceleration, *accel*, to the suspension data structure, *out_susp*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_susp	pointer to int	Standard
accel	REAL	/simnet/common/include/global/sim_types.h

Table 2.5-167: suspension_acceleration_is Information.

2.5.6.4 susp_init.c

(/simnet/release/src/vehicle/libsrc/libsusp/susp_init.c)

Includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"susp_loc.h"

2.5.6.4.1 suspension_uninit

This routine resets the suspension_inited field of the local suspension data structure to FALSE. If the suspension was not initialized to begin with, a message "PANIC -- tried to uninit non-inited workspace" is printed. The suspension is not really uninitialized; a variable is reset.

Parameters		
Parameter	Type	Where Typedef Declared
out_susp	pointer to int	Standard
Internal Variables		
Variable	Type	Where Typedef Declared
loc_susp	pointer to SUSPENSION	susp_loc.h

Table 2.5-168: suspension_uninit Information.

2.5.6.4.2 suspension_init

This routine initializes the suspension, pointing the local suspension data structure to the passed parameter, *out_susp*. If memory does not exist, the routine allocates this structure, then initializes some of the elements in the structure.

Parameters		
Parameter	Type	Where Typedef Declared
out_susp	pointer to pointer to int	Standard
Internal Variables		
Variable	Type	Where Typedef Declared
loc_susp	pointer to SUSPENSION	susp_loc.h

Calls	
Function	Where Described
dynamics_filter_init	Section 2.5.4.7.1
suspension_uninit	Section 2.5.6.4.1

Table 2.5-169: suspension_init Information.

2.5.6.5 susp_params.c
(/simnet/release/src/vehicle/libsrc/libsusp/susp_params.c)

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"libmatrix.h"
"susp_loc.h"

2.5.6.5.1 suspension_params

This routine sets up the suspension parameters. The suspension is modeled as a second order filter based on the natural frequency and damping ratio. The lever arm and angle limits are set.

rot_wn -- Rotational suspension natural frequency (radians)
rot_zeta -- Rotational suspension damping ratio
side_wn -- Side suspension natural frequency (radians)
side_zeta -- Side suspension damping ratio
lever_arm -- Meters
angle_lim -- Maximum angle limit ~9 degrees .7 m by 4.5 m
gun_force -- The force generated for rocking the gun
left -- Left side offset from rear wheel
right -- Right side offset from rear wheel

Parameters		
Parameter	Type	Where Typedef Declared
out_susp	pointer to int	Standard
rot_wn	REAL	/simnet/common/include/global/sim_types.h
rot_zeta	REAL	/simnet/common/include/global/sim_types.h
side_wn	REAL	/simnet/common/include/global/sim_types.h
side_zeta	REAL	/simnet/common/include/global/sim_types.h
lever_arm	REAL	/simnet/common/include/global/sim_types.h
angle_lim	REAL	/simnet/common/include/global/sim_types.h
gun_force	REAL	/simnet/common/include/global/sim_types.h
left	pointer to REAL	/simnet/common/include/global/sim_types.h
right	pointer to REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
vec_copy	Section 2.6.2.59.1

Table 2.5-170: suspension_params Information.

2.5.6.6 susp_simul.c

(/simnet/release/src/vehicle/libsrc/libsusp/susp_simul.c)

Includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"dynlib.h"
"susp_loc.h"
"libmatrix.h"

2.5.6.6.1 suspension

This routine is called on a tick by tick basis. Given a pointer to the suspension structure, the routine determines the location of the wheels and how high they are above the terrain. The dynamics package is not initialized. The rotation (forward to back rocking) is found as a product of the acceleration and level arm with a rotational input added from the force of the gun. Parameters and variables are represented as follows:

out_susp -- local suspension parameters structure
rear_wheel -- Location of rear wheels in world coordinates; it is assumed they are sitting on the terrain patch
h_to_w -- Hull to World coordinate transformation matrix
u_norm -- The normal to the terrain patch under the vehicle
aX,aY -- Temporary variables
rot_angle -- Rotation angle; Pitch of the suspension relative to the world of the vehicle
temp -- Temporary storage variable
left_side -- Left side point of the tank
right_side -- Right side point of the tank
forward -- Forward point of the tank

First, the offset in world coordinates is calculated. The heights in the plane are calculated from the unit normal. The forward vector and pitch angle are calculated. The states are updated and the new unit normal is formed relative to the orientation of the tank.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_susp	pointer to int	Standard
rear_wheel	VECTOR	/simnet/common/include/global/sim_types.h
h_to_w	T_MATRIX	/simnet/common/include/global/sim_types.h
u_norm	VECTOR	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
loc_susp	pointer to SUSPENSION	susp_loc.h
aX	register REAL	/simnet/common/include/global/sim_types.h
aY	register REAL	/simnet/common/include/global/sim_types.h
rot_angle	register REAL	/simnet/common/include/global/sim_types.h
temp	VECTOR	/simnet/common/include/global/sim_types.h
forward	VECTOR	/simnet/common/include/global/sim_types.h
left_side	VECTOR	/simnet/common/include/global/sim_types.h
right_side	VECTOR	/simnet/common/include/global/sim_types.h
rot_input	register REAL	/simnet/common/include/global/sim_types.h
left_input	register REAL	/simnet/common/include/global/sim_types.h
right_input	register REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
vec_mat_mul	Section 2.6.2.56.1
vec_sub	Section 2.6.2.65.1
vec_cross_prod	Section 2.6.2.66.1
dynamics_filter_update	Section 2.5.7.4.3
vec_normalize	Section 2.6.2.63.1

Table 2.5-171: suspension Information.

2.5.6.7 veh_init.c
(/simnet/release/src/vehicle/libsrc/libsusp/veh_init.c)

Includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"susp_loc.h"
"dynlib.h"
"libmatrix.h

2.5.6.7.1 suspension_veh_init

This routine finds the relative location in world coordinates of the rear wheels (left side and right side), assuming the wheels are sitting on the terrain patch. The offset is calculated in world coordinates, then heights in the plane from the unit normal are calculated. A vector pointing ahead is formed (forward). The pitch angle is calculated, and the states are initialized. Parameter and variable are represented as follows:

out_susp -- Local suspension parameters structure
rear_wheel -- Location of rear wheels in world coordinates
h_to_w -- Hull to World coordinate transformation matrix
u_norm -- The normal to the terrain patch under the vehicle
rot_angle -- Rotation angle; Pitch of suspension relative to the vehicle world
temp -- Temporary storage variable
left_side -- Left side point of the tank
right_side -- Right side point of the tank
forward -- Forward point of the tank

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_susp	pointer to int	Standard
rear_wheel	VECTOR	/simnet/common/include/global/sim_types.h
h_to_w	T_MATRIX	/simnet/common/include/global/sim_types.h
u_norm	VECTOR	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
loc_susp	register pointer SUSPENSION	susp_loc.h
aX	register REAL	/simnet/common/include/global/sim_types.h
aY	register REAL	/simnet/common/include/global/sim_types.h
rot_angle	register REAL	/simnet/common/include/global/sim_types.h
temp	VECTOR	/simnet/common/include/global/sim_types.h
forward	VECTOR	/simnet/common/include/global/sim_types.h
left_side	VECTOR	/simnet/common/include/global/sim_types.h
right_side	VECTOR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
vec_mat_mul	Section 2.6.2.56.1
vec_sub	Section 2.6.2.65.1
vec_cross_prod	Section 2.6.2.66.1
dynamics_filter_open	Section 2.5.7..4.2

Table 2.5-172: suspension_veh_init Information.

2.5.6.8 sus_loc.h

(/simnet/release/src/vehicle/libsrc/libsusp/susp_loc.h)

Includes:

"stdio.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"dynlib.h"

The SUSPENSION data structure type is defined, and contains the following fields:

ROT_WN	-- rotational suspension natural frequency (in radians)
ROT_ZETA	-- rotational suspension damping ratio
SIDE_WN	-- side suspension natural frequency (in radians)
SIDE_ZETA	-- side suspension damping ratio
LEVER_ARM	-- in meters
ANGLE_LIM	-- approximately 9 degrees, 0.7 meters by 4.5 meters
GUN_FORCE	-- force of firing the gun
rot_suspension	
right_suspension	
left_suspension	
l_offset	-- left side offset from rear wheel
r_offset	-- right side offset from rear wheel
gun_fired	-- raised when the gun is fired
turret_cos	
turret_sin	
veh_accel	
suspension_inited	

2.5.7 libdyn
(/simnet/release/src/vehicle/libsrc/libdyn [libdyn])

This library provides simple utilities for the creation and maintenance of second order filters, a first order lag function, integration of forces and torques to form accelerations, integration of accelerations to form velocities, and the calculation of inertias based on rotation rate and velocity. These facilities are used throughout the M1 simulation.

2.5.7.1 calc_inert.c
(/simnet/release/src/vehicle/libsrc/libdyn/calc_inert.c)

Includes:
"sim_types.h"
"sim_dfns.h"
"libmatrix.h"
"dyn_mass.h"

2.5.7.1.1 dynamics_calc_inertial_forces

This routine calculates gyroscopic torques and centrifugal forces according to the following algorithm:
$T = -w \times Iw = (Iw) \times w$
$R = -M\,(w \times v) = M\,(v \times w)$
Parameters are represented as follows:
massP -- mass properties structure
w -- angular velocities
v -- velocities
T -- resultant torque
R -- resultant force

Parameters		
Parameter	**Type**	**Where Typedef Declared**
massP	pointer to MASS_PROP	/simnet/release/src/libsrc/include/dyn_mass.h
w	pointer to REAL	/simnet/common/include/global/sim_types.h
v	pointer to REAL	/simnet/common/include/global/sim_types.h
T	VECTOR	/simnet/common/include/global/sim_types.h
R	VECTOR	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
Angular_momenturm	VECTOR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_vec_mul	Section 2.6.2.35.1

Table 2.5-173: dynamics_calc_inertial_forces Information.

2.5.7.2 calc_u.c
(/simnet/release/src/vehicle/libsrc/libdyn/calc_u.c)

Includes:
"sim_types.h"
"sim_dfns.h"

2.5.7.2.1 dynamics_calc_u

This routine integrates *udot* (an acceleration) to get *u* (a velocity). This routine is used to update the velocity given a previous velocity vector, a previous angular acceleration, and a previous linear acceleration. Parameters are represented as follows:

alpha -- angular acceleration
a -- acceleration
w -- angular velocity
v -- velocity

Parameters		
Parameter	**Type**	**Where Typedef Declared**
alpha	pointer to REAL	/simnet/common/include/global/sim_types.h
a	pointer to REAL	/simnet/common/include/global/sim_types.h
w	pointer to REAL	/simnet/common/include/global/sim_types.h
v	pointer to REAL	/simnet/common/include/global/sim_types.h

Table 2.5-174: dynamics_calc_u Information.

2.5.7.3 calc_udot.c
(/simnet/release/src/vehicle/libsrc/libdyn/calc_udot.c)

Includes:
"sim_types.h"
"sim_dfns.h"
"dyn_mass.h"
"libmatrix.h"

2.5.7.3.1 dynamics_calc_udot

This routine calculates the new linear and angular accelerations given the torque and force applied to a mass according to the following algorithms:

$alpha = I^{-T}$

$a = R / m$

Parameters are represented as follows:

massP -- mass properites structure
T -- generalized active torques
R -- generalized active forces
alpha -- angular acceleration
a -- acceleration

Parameters		
Parameter	**Type**	**Where Typedef Declared**
massP	pointer to MASS_PROP	/simnet/release/src/libsrc/include/dyn_mass.h
T	VECTOR	/simnet/common/include/global/sim_types.h
R	VECTOR	/simnet/common/include/global/sim_types.h
alpha	pointer to REAL	/simnet/common/include/global/sim_types.h
a	pointer to REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_vec_mul	Section 2.6.2.35.1
vec_scale	Section 2.6.2.64.1

Table 2.5-175: dynamics_calc_udot Information.

2.5.7.4 filter.c
(/simnet/release/src/vehicle/libsrc/libdyn/filter.c)

Includes:
"stdio.h"
"sys/types.h" (MASSCOMP only)
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"dynlib.h"

2.5.7.4.1 dynamics_filter_init

This routine allocates memory for a filter, setting a pointer to the filter.

Return Values		
Return Value	Type	Meaning
loc_filter	pointer to FILTER	a pointer to the allocated filter

Table 2.5-176: dynamics_filter_init Information.

2.5.7.4.2 dynamics_filter_open

This routine builds a second order filter, initializes values, and returns a pointer to the filter instance.

Parameters are represented as follows:

filterP -- the filter structure pointer
zeta -- the damping ratio
wn -- the natural frequency (rads/sec)
limit -- the maximum change from input (negative if there is none)
timinc -- the time increment
init -- the initial value of the filter

Variables are represented as follows:

a and *b* -- are of the form $1/((s+a)^2 + b^2)$
r -- radius on Z-plane
theta -- angle on Z-plane

Parameters		
Parameter	**Type**	**Where Typedef Declared**
filterP	pointer to FILTER	/simnet/release/src/libsrc/include/dynlib.h
zeta	REAL	/simnet/common/include/global/sim_types.h
wn	REAL	/simnet/common/include/global/sim_types.h
limit	REAL	/simnet/common/include/global/sim_types.h
timinc	REAL	/simnet/common/include/global/sim_types.h
init	REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
a	REAL	/simnet/common/include/global/sim_types.h
b	REAL	/simnet/common/include/global/sim_types.h
r	REAL	/simnet/common/include/global/sim_types.h
theta	REAL	/simnet/common/include/global/sim_types.h

Table 2.5-177: dynamics_filter_open Information.

2.5.7.4.3 dynamics_filter_update

Given an input value, this routine updates the filter and returns an output value.

Parameters are represented as follows:
fP -- the filter pointer
in -- the input value
in2 -- the second input (for forces)

The variable *out* represents the output at the current time step.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fP	pointer to register FILTER	/simnet/release/src/libsrc/include/dynlib.h
in	REAL	/simnet/common/include/global/sim_types.h
in2	REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
out	REAL	/simnet/common/include/global/sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
out	REAL	the output at the current time step

Table 2.5-178: dynamics_filter_update Information.

2.5.7.5 init.c
(/simnet/release/src/vehicle/libsrc/libdyn/init.c)

Includes:
"sim_types.h"
"sim_dfns.h"
"dyn_mass"

Procedure Declarations:
dump_mass()

2.5.7.5.1 dynamics_init

This routine initializes the mass properties matrix for body B.

Parameters are represented as follows:
massP -- mass properties structure
Mass -- mass of body B
I -- Inertia matrix of B about its center of mass

Parameters		
Parameter	**Type**	**Where Typedef Declared**
massP	pointer to MASS_PROP	/simnet/release/src/libsrc/include/dyn_mass.h
Mass	REAL	/simnet/common/include/global/sim_types.h
I	T_MATRIX	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_copy	Section 2.6.2.39.1
mat_inverse	Section 2.6.2.45.1

Table 2.5-179: dynamics_init Information.

2.5.7.5.2 dump_mass

This routine is used for setting certain printouts during debugging.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
massP	pointer to MASS_PROP	/simnet/release/src/libsrc/include/dyn_mass.h

Calls	
Function	**Where Described**
mat_dump	Section 2.6.2.41.1

Table 2.5-180: dump_mass Information.

2.5.7.6 lag.c
(/simnet/release/src/vehicle/libsrc/libdyn/lag.c)

Includes:
"sim_types.h"
"sim_dfns.h"

2.5.7.6.1 first_order_lag

This routine calculates a first order lag in order to update the vehicle position.

Parameters are represented as follows:
present_x -- the present value
target_x -- the target value
time_constant -- the time constant to the system

Parameters		
Parameter	**Type**	**Where Typedef Declared**
present_x	REAL	/simnet/common/include/global/sim_types.h
target_x	REAL	/simnet/common/include/global/sim_types.h
time_constant	REAL	/simnet/common/include/global/sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
present_x	REAL	the present value

Table 2.5-181: first_order_lag Information.

2.5.8 libkin
(/simnet/release/src/vehicle/libsrc/libkin [libkin])

This library maintains the kinematic state of the M1 and M2 from update data provided by vehicle specific code. There routines are usedto move the vehicle forward, turn the vehicle, calculate the direction cosine matrix of the vehicle, provide the square of the range from another point to the vehicle, and it provides access routines for all the internal kinematics information. It uses the unit normal vector for the terrain patch the vehicle is on, as provided by the libbigwheel library.

2.5.8.1 hull_info.c
(/simnet/release/src/vehicle/libsrc/libkin/hull_info.c)

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"kin_loc.h"

2.5.8.1.1 kinematics_get_w_to_h

This routine returns the world to hull transformation matrix. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Return Values		
Return Value	**Type**	**Meaning**
NULL	T_MAT_PTR	null pointer
(((HULL_INFO*)out_kinemat)->world_to_hull)	T_MAT_PTR	the world to hull transformation matrix

Table 2.5-182: kinematics_get_w_to_h Information.

2.5.8.1.2 kinematics_get_h_to_w

This routine returns the hull to world transformation matrix. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Return Values		
Return Value	**Type**	**Meaning**
NULL	T_MAT_PTR	null pointer
(((HULL_INFO*)out_kinemat)->hull_to_world)	T_MAT_PTR	the hull to world transformation matrix

Table 2.5-183: kinematics_get_h_to_w Information.

2.5.8.1.3 kinematics_get_h_to_o

This routine returns the hull to origin vector. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Return Values		
Return Value	**Type**	**Meaning**
NULL	pointer to REAL	null pointer
(((HULL_INFO*)out_kinemat)->hull_to_origin)	pointer to REAL	hull to origin vector

Table 2.5-184: kinematics_get_h_to_o Information.

2.5.8.1.4 kinematics_get_o_to_h

This routine returns the origin to hull vector. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Return Values		
Return Value	**Type**	**Meaning**
NULL	pointer to REAL	null pointer
(((HULL_INFO*)out_kinemat)->origin_to_hull)	pointer to REAL	origin to hull vector

Table 2.5-185: kinematics_get_o_to_h Information.

2.5.8.1.5 kinematics_get_u_norm

This routine returns the unit normal through the support plane. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Return Values		
Return Value	**Type**	**Meaning**
NULL	pointer to REAL	null pointer
(((HULL_INFO*)out_kinemat)->unit_normal)	pointer to REAL	the unit normal vector through the support plane

Table 2.5-186: kinematics_get_unit_normal Information.

2.5.8.1.6 kinematics_get_velocity

This routine returns the vehicle velocity in world coordinates in meters per second. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
loc_kinemat	pointer to HULL_INFO	kin_loc.h
Return Values		
Return Value	**Type**	**Meaning**
NULL	pointer to REAL	null pointer
veh_veloc	pointer to REAL	velocity of the vehicle

Table 2.5-187: kinematics_get_velocity Information.

2.5.8.1.7 kinematics_get_d_pos

This routine returns the change in position in world coordinates per DELTA_T. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Return Values		
Return Value	**Type**	**Meaning**
NULL	pointer to REAL	null pointer
(((HULL_INFO*)out_kinemat)->delta_position)	pointer to REAL	change in position in world coordinates

Table 2.5-188: kinematics_get_d_pos Information.

2.5.8.1.8 kinematics_get_slope_ind

This routine determines the hull direction and hull slope values. *hull_dir* represents the ange which characterizes the orientation of the hull in radians. *cos_hull_slope* represents the cosine of the angle from straight up. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
hull_dir	pointer to REAL	/simnet/common/include/global/sim_types.h
cos_hull_slope	pointer to REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
loc_kin	pointer to HULL_INFO	kin_loc.h
u_norm	pointer to REAL	/simnet/common/include/global/sim_types.h
temp_hull_dir	REAL	/simnet/common/include/global/sim_types.h
Calls		
Function	**Where Described**	
eq	Macro defined in /simnet/common/include/global/sim_macros.h	

Table 2.5-189: kinematics_get_slope_ind Information.

2.5.8.3 kin_init.c

(/simnet/release/src/vehicle/libsrc/libkin/kin_init.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"kin_loc.h"
"libkin.h"
"libbigwheel.h"

2.5.8.3.1 kinematics_uninit

This routine sets the locations to zero when the simulator is deactivated. This is used as a debugging tool. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Calls		
Function	**Where Described**	
kinematics_pos_init	Section 2.5.8.13.1	

Table 2.5-190: kinematics_uninit Information.

2.5.8.3.2 kinematics_init

This routine initializes the work space and fills in the initial parameters. Memory is allocated and a pointer to char is returned. That pointer is cast at a pointer to the HULL_INFO structure. The pointers are allocated and filled in. veh_kin is cast as a pointer to an int in order to insulate the HULL_INFO structure from the world outside the kinematics library.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_kin	pointer to pointer to int	Standard
veh_bigwh	pointer to pointer to int	Standard
veh_susp	pointer to pointer to int	Standard
veh_terr	pointer to pointer to int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
loc_kinemat	pointer to HULL_INFO	kin_loc.h
Calls		
Function	**Where Described**	
bigwheel_init	Section 2.5.10.1.2	
kinematics_uninit	Section2.5.8.3.1	

Table 2.5-191: kinematics_init Information.

2.5.8.4 kin_loc.c

(/simnet/release/src/vehicle/libsrc/libkin/kin_loc.c)

This file contains stuff

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"

Defines:

MAX_U_N_CHANGE
MAX_VELOC_CHANGE
MAX_HEADING_CHANGE

Declared:

RVA_U_NORM_CHECK
RVA_VELOC_CHECK
RVA_HEADING_CHECK

2.5.8.5 kin_loc.h

(/simnet/release/src/vehicle/libsrc/libkin/kin_loc.h)

The following functions are declared:

kinematics_set_local_kinematics ()
kinematics_vehicle_init()

The following constants are declared:

RVA_U_NORM_CHECK;
RVA_VELOC_CHECK;
RVA_HEADING_CHECK;

The kinematics_info structure is declared. The HULL_INFO stucture is instantiated.

2.5.8.6 kin_simul.c

(/simnet/release/src/vehicle/libsrc/libkin/kin_simul.c)

This file contains the primary kinematics simulaiton routine.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"kin_loc.h"
"libkin.h"
"bigwheel.h"
"libterrain.h"

2.5.8.6.1 kinematics_simul

Kinematics_simul() is the routine called every tick to perform various kinematics functions (**kinematics_set_local_kinematics**()). If kinematics has been initialized, then everything proceeds as normal. However, kinematics cannot be initialized until a valid patch of local terrain has been received. If kinematics has yet to be initialized, then a check is made to see if a terrain patch has been received recently. If so, then an attempt is made to initialize kinematics. However, if there is a problem with the terrain patch (incomplete coverage,etc.) then **bigwheel_init_support_plane**() will be unable to provide a valid unit normal, and kinematics remains uninitialized. When the next terrain patch is received, it tries again. After three unsuccessful tries, a panic message will be printed. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
loc_kin	pointer to HULL_INFO	kin_loc.h
Calls		
Function	**Where Described**	
kinematics_set_local_kinematics	Section 2.5.8.9.1	
terrain_lt_inited	Section 2.5.11.4.1	
bigwheel_init_support_plane	Section 2.5.10.7.1	
kinematics_vehicle_init	Section 2.5.8.13.2	
terrain_uninit	Section 2.5.11.7	

Table 2.5-192: kinematics_simul Information.

2.5.8.7 move_veh.c

(/simnet/release/src/vehicle/libsrc/libkin/move_veh.c)

This file contains a routine which moves the vehicle forward by the indicated increment.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"kin_loc.h"
"libkin.h"
"bigwheel.h"
"libmatrix.h"

2.5.8.7.1 kinematics_move_vehicle

This routine moves a vehicle forward by the indicated increment, *inc*. Check first to see if the vehicle can move. Move the vehicle along the Y axis in its own hull coordinates by the negation of the increment. Update the values in *o_to_h* and *d_pos*.

out_kinemat is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer ot int	Standard
inc	register REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
loc_kin	pointer to HULL_INFO	kin_loc.h
o_to_h	pointer to register REAL	/simnet/common/include/global/sim_types.h
h_to_o	pointer to register REAL	/simnet/common/include/global/sim_types.h
d_pos	pointer to register REAL	/simnet/common/include/global/sim_types.h
h_to_w	pointer to register T_MAT_PTR	/simnet/common/include/global/sim_types.h
Calls		
Function	**Where Described**	
eq	Macro defined in /simnet/common/include/global/sim_macros.h	
vec_mat_mul	Section 2.6.2.56.1	
vec_scale	Section 2.6.2.64.1	
vec_init	Section 2.6.2.61.1	

Table 2.5-193: kinematics_move_vehicle Information.

2.5.8.8 p_c_sines.c

(/simnet/release/src/vehicle/libsrc/libkin/p_c_sines.c)

This file contains routines which return the sime and cosine for the hull's pitch and cant.

Includes:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"kin_loc.h"

Defines:

CANT_SIN
PITCH_SIN
PITCH_COS
CANT_COS

2.5.8.5.1 kinematics_cant_cos

This routine returns the cosine of the angle at which the hull is canted. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Return Values		
Return Value	**Type**	**Meaning**
0.0	REAL	no data available
(((HULL_INFO) out_kinemat)->CANT_COS)	REAL	cosine of angle of cant

Table 2.5-194: kinematics_cant_cos Information.

2.5.8.5.2 kinematics_pitch_cos

This routine returns the cosine of the angle at which the hull is pitched. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Return Values		
Return Value	**Type**	**Meaning**
0.0	REAL	no data available
(((HULL_INFO*) out_kinemat)->PITCH_COS)	REAL	cosine of angle of pitch

Table 2.5-195: kinematics_pitch_cos Information.

2.5.8.5.3 kinematics_cant_sin

This routine returns the sine of the angle at which the hull is canted. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cas' : a pointer into the HULL_INFO structure. This allows the information in the H. ˙LL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Return Values		
Return Value	**Type**	**Meaning**
0.0	REAL	no data available
(((HULL_INFO*) out_kinemat)->CANT_SIN)	REAL	sine of cant angle

Table 2.5-196: kinematics_cant_sin Information.

2.5.8.5.1 kinematics_pitch_sin

This routine returns the sine of the angle at which the hull is pitched. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Return Values		
Return Value	**Type**	**Meaning**
0.0	REAL	no data available
(((HULL_INFO*) out_kinemat)->PITCH_SIN)	REAL	sine of pitch angle

Table 2.5-197: kinematics_pitch_sin Information.

2.5.8.9 set_loc_kin.c

(/simnet/release/src/vehicle/libsrc/libkin/set_loc_kin.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"kin_loc.h"
"libkin.h"
"bigwheel.h"
"libmatrix.h"
"libterrain.h"

Declared:

get_orient_vecs()
kinematics_fix_matrix()

2.5.8.9.1 kinematics_set_local_kinematics

This routine computes the transform matrix from the previous hull matrix and the new unit normal for the local tank. If the unit normal has not changed since the last tick (which would happen if the vehicle was stopped and stable, or moving on flat ground and stable), then local_kinematics is not run. This routine is called at least twice a second to ensure that the vehicle does not drift underground while traveling on a level surface when the vehicle itself is not level.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
loc_kin	pointer to HULL_INFO	kin_loc.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
d_orient_mat	T_MATRIX	/simnet/common/include/global/sim_types.h
temp_matrix	register T_MAT_PTR	/simnet/common/include/global/sim_types.h
w_to_h	register T_MAT_PTR	/simnet/common/include/global/sim_types.h
a	register REAL	/simnet/common/include/global/sim_types.h
b	register REAL	/simnet/common/include/global/sim_types.h
abs_b	register REAL	/simnet/common/include/global/sim_types.h
c	register REAL	/simnet/common/include/global/sim_types.h
abs_c	register REAL	/simnet/common/include/global/sim_types.h
max_b_c	register REAL	/simnet/common/include/global/sim_types.h
temp	register REAL	/simnet/common/include/global/sim_types.h
sqr_temp	register REAL	/simnet/common/include/global/sim_types.h
denom	register REAL	/simnet/common/include/global/sim_types.h
new_height	register REAL	/simnet/common/include/global/sim_types.h
h_to_o	pointer to register REAL	/simnet/common/include/global/sim_types.h
u_norm	pointer to register REAL	/simnet/common/include/global/sim_types.h
u	VECTOR	/simnet/common/include/global/sim_types.h
v	VECTOR	/simnet/common/include/global/sim_types.h
w	VECTOR	/simnet/common/include/global/sim_types.h
force_local_kin	static int	Standard

Calls	
Function	**Where Described**
bigwheel_set_support_plane	Section 2.5.10.8.1
vec_dot_prod	Section 2.6.2.54.1
get_orient_vecs	Section 2.5.8.9.3
max	Macro defined in /simnet/common/include/global/sim_macros.h
eq	Macro defined in /simnet/common/include/global/sim_macros.h
square	Macro defined in /simnet/common/include/global/sim_macros.h
mat_mat_mul	Section 2.6.2.32.1
mat_transpose	Section 2.6.2.52.1
terrain_calc_elev	Section 2.5.11.1.1
kinematics_fix_matrix	Section 2.5.8.9.2
vec_mat_mul	Section 2.6.2.56.1

Table 2.5-198: kinematics_set_local_kinematics Information.

2.5.8.9.2 kinematics_fix_matrix

This routine fixes a world to hull matrix (*w_to_h*), its transverse (*h_to_w*), and hull to origin (*h_to_o*) vector to prevent disasters. It also fixes the origin_to_hull (o-*to_h*) vector. This routine is called to reorthonormalize the matrices.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w_to_h	register T_MATRIX	/simnet/common/include/global/sim_types.h
h_to_w	T_MATRIX	/simnet/common/include/global/sim_types.h
h_to_o	register VECTOR	/simnet/common/include/global/sim_types.h
o_to_h	VECTOR	/simnet/common/include/global/sim_types.h
Calls		
Function	**Where Described**	
mat_fix_matrix	Section 2.6.2.30.1	
mat_transpose	Section 2.6.2.51.1	
vec_mat_mul	Section 2.6.2.56.1	

Table 2.5-199: kinematics_fix_matrix Information.

2.5.8.9.3 get_orient_vecs

This routine gets the orientation vectors.

Parameters		
Parameter	Type	Where Typedef Declared
loc_kin	pointer to HULL_INFO	kin_loc.h
u_ptr	register VECTOR	/simnet/common/include/global/sim_types.h
v_ptr	register VECTOR	/simnet/common/include/global/sim_types.h
w_ptr	register VECTOR	/simnet/common/include/global/sim_types.h
Internal Variables		
Internal Variable	Type	Where Typedef Declared
t_real	register pointer to REAL	/simnet/common/include/global/sim_types.h

Table 2.5-200: get_orient_vecs Information.

2.5.8.10 sqr_range.c
(/simnet/release/src/vehicle/libsrc/libkin/sqr_range.c)

This file contains a routine which computes the square of the range between a point and the vehicle.

Includes:
"stdio,h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"kin_loc.h"
"libkin.h"
"libmatrix.h"

2.5.8.10.1 kinematics_range_squared

This routine computes the square of the range between the last position and the current position in three directions. This routine is called from "librva". *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
p2	pointer to register double	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
o2hp	register pointer to double	Standard
delta_x	register double	Standard
delta_y	register double	Standard
delta_z	register double	Standard
Return Values		
Return Value	**Type**	**Meaning**
0.0	REAL	no data available
((delta_x*delta_x)+(delta_y*delta_y)*(delta_z*delta_z))	REAL	the square of the range between two points

Table 2.5-201: kinematics_range_squared Information.

2.5.8.11 turn_veh.c

(/simnet/release/src/vehicle/libsrc/libkin/turn_veh.c)

This file contains a routine which turns the vehicle in its local coordinate system.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"kin_loc.h"
"libkin.h"
"libmatrix.h"

2.5.8.11.1 kinematics_turn_vehicle

This routine turns the vehicle in its local coordinate system. Since the vehicle turns less than 0.1 radians per DELTA_T seconds, the approximation x=sin(x) is used. *angle* is the turn angle. A turn to the left is denoted as a positive angle, and a turn to the right is denoted as a negative angle. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
angle	register REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
loc_kin	pointer to HULL_INFO	kin_loc.h
rot_matrix	T_MATRIX	/simnet/common/include/global/sim_types.h
r_mat	register T_MAT_PTR	/simnet/common/include/global/sim_types.h
o_to_h	pointer to register REAL	/simnet/common/include/global/sim_types.h
h_to_o	pointer to register REAL	/simnet/common/include/global/sim_types.h
w_to_h	register T_MAT_PTR	/simnet/common/include/global/sim_types.h
h_to_w	register T_MAT_PTR	/simnet/common/include/global/sim_types.h
Calls		
Function	**Where Described**	
mat_rot_init2	Section 2.6.2.33.1	
mat_mat_mul	Section 2.6.2.32.1	
mat_transpose	Section 2.6.2.51.1	
vec_mat_mul	Section 2.6.2.56.1	

Table 2.5-202: kinematics_turn_vehicle Information.

2.5.8.12 update.c
(/simnet/release/src/vehicle/libsrc/libkin/update.c)

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"kin_loc.h"
"libkin.h"
"libmatrix.h"

2.5.8.12.1 kinematics_update_rva

This routine is used to save local kinematics data. *out_kinemat* is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
loc_kin	pointer to HULL_INFO	kin_loc.h
Calls		
Function	**Where Described**	
vec_copy	Section 2.6.2.59.1	

Table 2.5-203: kinematics_update_rva Information.

2.5.8.13 veh_init.c
(/simnet/release/src/vehicle/libsrc/libkin/veh_init.c)

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"kin_loc.h"
"libkin.h"
"libmatrix.h"
"bigwheel.h"
"libterrain.h"

2.5.8.13.1 kinematics_pos_init

This routine is called when the vehicle is initialized. The initial position and heading of the tank (*x*, *y*, and *yaw*) are passed in. If x and y are divisible by 125, 0.5 is added to eliminate ambiguity about which terrain patch the vehicle is on. Note that the actual initialization of the vehicle is not done here, but in the routine **kinematics_vehicle_init().**

out_kinemat is a global variable which is used to access the HULL_INFO structure from outside the kinematics library. It is passed into this routine as the primary parameter and is a pointer to an int. Within this routine, *out_kinemat* is cast as a pointer into the HULL_INFO structure. This allows the information in the HULL_INFO structure to be insulated from the rest of the simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_kinemat	pointer to int	Standard
x	register REAL	/simnet/common/include/global/sim_types.h
y	register REAL	/simnet/common/include/global/sim_types.h
yaw	register REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
loc_kin	*pointer to HULL_INFO*	*kin_loc.h*

Calls	
Function	**Where Described**
kinematics_vehicle_init	Section 2.5.8.13.2
vec_copy	Section 2.6.2.59.1

Table 2.5-204: kinematics_pos_init Information.

2.5.8.13.2 kinematics_vehicle_init

This routine initializes the transform matrix, its inverse, its unit normal vector, and the hull_to_origin vector of a vehicle, given its desired x-y location and heading in world coordinates.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
loc_kin	pointer to HULL_INFO	kin_loc.h
x	register REAL	/simnet/common/include/global/sim_types.h
y	register REAL	/simnet/common/include/global/sim_types.h
yaw	register REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
w_to_h	register T_MAT_PTR	/simnet/common/include/global/sim_types.h
h_to_w	register T_MAT_PTR	/simnet/common/include/global/sim_types.h
h_to_o	pointer to register REAL	/simnet/common/include/global/sim_types.h
o_to_h	pointer to register REAL	/simnet/common/include/global/sim_types.h
u_norm	pointer to register REAL	/simnet/common/include/global/sim_types.h
o_mat	T_MATRIX	/simnet/common/include/global/sim_types.h
temp_norm	VECTOR	/simnet/common/include/global/sim_types.h
temp1	VECTOR	/simnet/common/include/global/sim_types.h
temp2	VECTOR	/simnet/common/include/global/sim_types.h
temp3	VECTOR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_rot_init	Section 2.6.2.47.1
terrain_calc_elev	Section 2.5.11.1.1
vec_scale	Section 2.6.2.64.1
vec_mat_mul	Section 2.6.2.56.1
mat_transpose	Section 2.6.2.51.1
bigwheel_init_support_plane	Section 2.5.10.7.1
vec_cross_prod	Section 2.6.2.66.1
mat_mat_mul	Section 2.6.2.32.1

Table 2.5-205: kinematics_vehicle_init Information.

2.5.9 libhull
(/simnet/release/src/vehicle/libsrc/libhull [libhull])

This library initializes the hull of a vehicle by allocating memory for the kinematics, terrain, suspension, and bigwheel state data. It is called by main.c in /simnet/release/vehicle/libsrc/libmain.

2.5.9.1 hull_init.c
(/simnet/release/src/vehicle/libsrc/libhull/hull_init.c)

This file contains routines which initialize and uninitialize the hull of a vehicle.

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libkin.h"
"libsusp.h"
"libhull.h"
"libbigwheel.h"
"libterrain.h"

2.5.9.1.1 hull_init

This routine initializes the hull of a vehicle by allocating memory for the kinematics, terrain, suspension, and bigwheel state data.

Calls	
Function	Where Described
kinematics_init	Section 2.5.8.3.2

Table 2.5-206: hull_init Information.

2.5.9.1.2 hull_uninit

This routine uninitializes the hull of a vehicle by deallocating memory for the kinematics, terrain, suspension, and bigwheel state data.

Calls	
Function	Where Described
bigwheel_uninit	Section 2.5.10.1.1
suspension_uninit	Section 2.5.6.4.1
terrain_uninit	Section 2.5.11.7
kinematics_uninit	Section 2.5.8.3.1

Table 2.5-207: hull_uninit Information.

2.5.9.2 hull_loc.c

(/simnet/release/src/vehicle/libsrc/libhull/hull_loc.c)

The following pointers are declared and initialized:

veh_kinematics
veh_bigwheel
veh_suspension
veh_terrain

2.5.10 libbigwh
(/simnet/release/src/vehicle/libsrc/libbigwh [libbigwh])

The support plane of the vehicle is determined by taking three points under the tank (right front, left front, and rear -- like a backward child's bigwheel) and calculating a unit normal to that plane. This library calculates this normal, and passes it onto the kinematics library. The bigwheel library also registers collisions with other objects on the terrain and informs the kinematics library. The M-1 tank is modeled as the three wheels of a child's "Bigwheel". The three points of contact with the terrain define a support plane for the tank. The unit normal of this support plane is computed and passed on to the kinematics code.

2.5.10.1 bigwh_init.c
(/simnet/release/src/vehicle/libsrc/libbigwh/bigwh_init.c)

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"bigwh_loc.h"
"libterrain.h"
"libsusp.h"

2.5.10.1.1 bigwheel_uninit

This routine uninitializes the bigwheel data structure by zeroing out the structure and setting the number of collisions to zero.

Parameters		
Parameter	Type	Where Typedef Declared
out_bigwh	pointer to int	Standard
Internal Variables		
Variable	Type	Where Typedef Declared
loc_bigwh	pointer to BIGWHEEL	Section 2.5.10.2

Table 2.5-208: bigwheel_uninit Information.

2.5.10.1.2 bigwheel_init

This routine is called by **kinematics_init()** to initialize the local vehicle's bigwheel workspace. The local data structure is allocated.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_bigwheel	pointer to pointer to int	Standard
out_suspension	pointer to pointer to int	Standard
out_terrain	pointer to pointer to int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
loc_bigwh	pointer to BIGWHEEL	Section 2.5.10.2
Calls		
Function	**Where Described**	
suspension_init	Section 2.5.6.4.2	
terrain_init	Section 2.5.11.7	
bigwheel_uninit	Section 2.5.10.1.1	

Table 2.5-209: bigwheel_init Information.

2.5.10.2 bigwh_loc.h

(/simnet/release/src/vehicle/libsrc/libbigwh/bigwh_loc.h)

Includes
"bigwheel.h"

Defines
BIGWHEEL_DEBUG

The following are declared as external:
bigwheel_calc_unit_normal();
collision_detected();
collision_cleared();

The following structure is defined:
BIGWHEEL;

2.5.10.3 calc_u_norm.c
(/simnet/release/src/vehicle/libsrc/libbigwh/calc_u_norm.c)

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"bigwh_loc.h"
"libmatrix.h"

2.5.10.3.1 bigwheel_calc_unit_normal

This routine calculates the unit normal vector of the plane defined by three the points of support.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
wheels[3]	register VECTOR	/simnet/common/include/global/sim_types.h
result	register VECTOR	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
v1	VECTOR	/simnet/common/include/global/sim_types.h
v2	VECTOR	/simnet/common/include/global/sim_types.h
v1_ptr	pointer to register REAL	/simnet/common/include/global/sim_types.h
v2_ptr	pointer to register REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
vec_sub	Section 2.6.2.65.1
vec_cross_prod	Section 2.6.2.66.1
vec_normalize	Section 2.6.2.63.1

Table 2.5-210: bigwheel_calc_unit_normal Information.

2.5.10.4 chk_coll.c

(/simnet/release/src/vehicle/libsrc/libbigwh/chk_coll.c)

Includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"bigwh_loc.h"

2.5.10.4.1 collision_left_collision

This routine is called by kinematics to check to determine if a collision has occurred on the left side of the vehicle.

Parameters		
Parameter	Type	Where Typedef Declared
out_bigwheel	pointer to int	Standard
Return Values		
Return Value	Type	Meaning
FALSE	int	no collision on left side
TRUE	int	collision occurred on left side

Table 2.5-211: collision_left_collision Information.

2.5.10.4.2 collision_right_collision

This routine is called by kinematics to determine if a collision has occurred on the right side of the vehicle.

Parameters		
Parameter	Type	Where Typedef Declared
out_bigwheel	pointer to int	Standard
Return Values		
Return Value	Type	Meaning
FALSE	int	no collision on right side
TRUE	int	collision occurred on right side

Table 2.5-212: collision_right_collision Information.

2.5.10.4.3 collision_rear_collision

This routine is called by kinematics to determine if a collision has occurred on the rear of the vehicle.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_bigwheel	pointer to int	Standard
Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	no collision on rear
TRUE	int	collision occurred on rear

Table 2.5-213: collision_rear_collision Information.

2.5.10.5 coll_init.c

(/simnet/release/src/vehicle/libsrc/libbigwh/coll_init.c)

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"bigwh_loc.h"

2.5.10.5.1 collision_init

This routine is called by the vehicle specific code in order to set up the failure routine to be called when a collision occurs. Currently, only the turret drive systems fail in a collision.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_bigwheel	pointer to int	Standard
failure_rtn	PFI	/simnet/common/include/global/sim_types.h

Table 2.5-214: collision_init Information.

2.5.10.6 collision.c
(/simnet/release/src/vehicle/libsrc/libbigwh/collision.c)

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"pro_sim.h"
"obj_type.h"
"libkin.h"
"librva.h"
"libevent.h"
"libevent.h"
"bigwh_loc.h"
"bigwheel.h"

Defines:
COLL_RANGE

Declarations:
RANGE_SQRD

2.5.10.6.1 collision_check_veh_coll_at

This routine is called by librva during the tick by tick processing and serves two purposes:

1) When another vehicle is close, this routine checks to see whether a collision has occurred, or whether it was just a close miss. To check for a collision, the distance (in the x,y plane) is calculated between each of our three bigwheels and the closest vehicle's center of mass. If the closest distance is less than the collision range, then a collision has occurred. Collisions may not occur between vehicles at different heights (e.g., an airplane and a truck). The routine also check to make sure that the object collided with was not a missile.

2) When we receive a collision packet from another vehicle simulation, this routine calculates from which direction we were hit. Note that if a collision packet is received from another vehicle, we confirm a collision whether or not we actually detected it (the *confirmed_hit* flag is set). If nothing is in range after a collision packet has been received, we assume that the other vehicle backed off and we clear the collision.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_bigwheel	pointer to int	Standard
confirmed_hit	int	Standard
hash_id	int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
loc_bigwh	pointer to BIGWHEEL	Section 2.5.10.2
other_tank_pos	VECTOR	/simnet/common/include/global/sim_types.h
loc	pointer to REAL	/simnet/common/include/global/sim_types.h
o_t_pos	pointer to register REAL	/simnet/common/include/global/sim_types.h
compute_sqr_range()	REAL	/simnet/common/include/global/sim_types.h
rear_r	REAL	/simnet/common/include/global/sim_types.h
left_r	REAL	/simnet/common/include/global/sim_types.h
right_r	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
rva_get_object_type	Section 2.5.12.5.1
rva_get_veh_loc	Section 2.5.12.8.1
compute_sqr_range	Section 2.5.10.9.1
collision_detected	Section 2.5.10.6.3
collision_cleared	Section 2.5.10.6.2

Table 2.5-215: collison_check_veh_coll_at Information.

2.5.10.6.2 collision_cleared

This routine clears a collision if no vehicle is in range after a collision packet has been received. Note that in theory, a vehicle can collide with more than one object. This routine checks to see if the *cause* of the clear is equal to the *cause* of the collision in case you are still collided with something else.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
loc_bigwh	pointer to BIGWHEEL	Section 2.5.10.2
coll_dir	int	Standard
cause	int	Standard

Table 2.5-216: collision_cleared Information.

2.5.10.6.3 collision_detected

This routine is called when a collision is detected. The routine ignores multiple collisions on the same side of the vehicle, therefore the network may not necessarily be called to send out a collision packet for every collision. Note that the newly collided vehicle should still send a packet; there should always be a record of the collision.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
loc_bigwh	pointer to BIGWHEEL	Section 2.5.10.2
coll_dir	int	Standard
cause	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
coll_eventid	int	Standard
Calls		
Function	**Where Described**	
event_get_eventid	Section 2.6.9.1.1	

Table 2.5-217: collision_detected Information.

2.5.10.6.4 collision_forget_about

This routine is called to clear away a collision if the vehicle you collided with is deactivated. The collision is cleared and your vehicle forgets that it was in a collision.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_bigwheel	pointer to int	Standard
hash_id	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
loc_bigwh	pointer to BIGWHEEL	Section 2.5.10.2

Table 2.5-218: collision_forget_about Information.

2.5.10.7 init_suppt.c

(/simnet/release/src/vehicle/libsrc/libbigwh/init_suppt.c)

Includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"bigwh_loc.h"
"bigwheel.h"
"libmatrix.h"
"libterrain.h"
"libsusp.h"

Procedure Declarations:

bigwh_init_height()

2.5.10.7.1 bigwheel_init_support_plane

When setting up the simulation, the first chunk of terrain is necessary to compute the orientation. This routine is called by **kinematics_vehicle_init()** to get an initial value for the unit normal, *u_norm*. If terrain coverage is incomplete, then the vehicle should not be initialized, so this routine returns FALSE. If terrain coverage is complete, then the unit normal is initialized and this routine returns TRUE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_bigwheel	pointer to int	Standard
h_to_w	register T_MATRIX	/simnet/common/include/global/sim_types.h
h_to_o	register VECTOR	/simnet/common/include/global/sim_types.h
u_norm	register VECTOR	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
loc_bigwh	pointer to BIGWHEEL	Section 2.5.10.2

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	unit normal is initialized
FALSE	int	incomplete terrain coverage

Calls	
Function	**Where Described**
bigwh_init_height	Section 2.5.10.7.2
bigwheel_cal_unit_normal	Section 2.5.10.3.1
suspension_veh_init	Section 2.5.6.7.1

Table 2.5-219: bigwheel_init_support_plane Information.

2.5.10.7.2 bigwheel_init_height

This routine called from **bigwheel_init_support_plane()** in order to get the initial height under each wheel. The routine makes a call to **terrain_calc_elev()** to calculate the elevation under each wheel; the height is pointed to in the variable *g_loc*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
loc_bigwh	pointer to BIGWHEEL	Section 2.5.10.2
wheel_num	int	Standard
h_to_o	VECTOR	/simnet/common/include/global/sim_types.h
h_to_w	T_MATRIX	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
g_loc	pointer to register REAL	Standard

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	terrain coverage complete
FALSE	int	terrain coverage not complete

Calls	
Function	**Where Described**
vec_sub	Section 2.6.2.65
vec_mat_mul	Section 2.6.2.56.1
terrain_calc_elev	Section 2.5.11.1.1

Table 2.5-220: bigwh_init_height Information.

2.5.10.8 set_suppt.c

(/simnet/release/src/vehicle/libsrc/libbigwh/set_suppt.c)

Include:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"bigwh_loc.h"
"libterrain.h"
"libmatrix.h"
"libsusp.h"
"bigwheel.h"

Procedure Declarations:

reg_gnd_wheel()
get_height_under_wheel()

2.5.10.8.1 bigwheel_set_support_plane

This routine is called each tick to get the unit normal, *u_norm*, of the hull to world matrix, *h_to_w*, and hull to origin vector, *h_to_o*, for the vehicle. This information is used to cant the vehicle to one side if a track is thrown.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_bigwheel	pointer toint	Standard
h_to_w	register T_MATRIX	/simnet/common/include/global/sim_types.h
h_to_o	register VECTOR	/simnet/common/include/global/sim_types.h
u_norm	register VECTOR	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
loc_bigwh	pointer to BIGWHEEL	Section 2.5.10.2
height	register REAL	/simnet/common/include/global/sim_types.h
wheel_down	int	Standard

Calls	
Function	**Where Described**
reg_gnd_wheel	Section 2.5.10.8.2
bigwheel_cal_unit_normal	Section 2.5.10.3.1
suspension	Section 2.5.6.6.1

Table 2.5-221: bigwheel_set_support_plane Information.

2.5.10.8.2 reg_gnd_wheel

The regular ground wheel routine checks for collisions against the terrain by getting the height under each wheel. If the height return is greater than the location and the track offset, the routine calls **collision_detected()**. This routine also allows the vehicle to continue moving at the same angle if a piece of terrain is missing.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
loc_bigwh	pointer to BIGWHEEL	Section 2 5.10.2
wheel_num	int	Standard
h_to_o	VECTOR	/simnet/common/include/global/sim_types.h
h_to_w	T_MATRIX	/simnet/common/include/global/sim_types.h
track_offset	REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
height	register REAL	/simnet/common/include/global/sim_types.h
loc_wheel	register REAL	/simnet/common/include/global/sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	The unit normal is out of range
TRUE	int	The unit normal is acceptable

Calls	
Function	**Where Described**
get_height_under_wheel	Section 2.5.10.8.3
collision_detected	Section 2.5.10.6.3
collision_cleared	Section 2.5.10.6.2

Table 2.5-222: reg_gnd_wheel Information.

2.5.10.8.3 get_height_under_wheel

This routine returns the height of the supporting terrain under the specified wheel.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
loc_bigwh	pointer to BIGWHEEL	Section 2.5.10.2
wheel_num	int	Standard
h_to_o	VECTOR	/simnet/common/include/global/sim_types.h
h_to_w	T_MATRIX	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
loc_wheel	pointer to register REAL	/simnet/common/include/global/sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
height	REAL	the height under the wheel

Calls	
Function	**Where Described**
vec_sub	Section 2.6.2.65
vec_mat_mul	Section 2.6.2.56.1
terrain_calc_elev	Section 2.5.11.1.1

Table 2.5-223: get_height_under_wheel Information.

2.5.10.9 sqr_range.c
(/simnet/release/src/vehicle/libsrc/libbigwh/sqr_range.c)

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"bigwh_loc.h"
"bigwheel.h"

2.5.10.9.1 compute_sqr_range

This routine checks the distance between two points standard and returns the two-dimensional (x-y) distance squared.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v1	pointer to register REAL	/simnet/common/include/global/sim_types.h
v2	pointer to register REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
range	register REAL	/simnet/common/include/global/sim_types.h
dist	register REAL	/simnet/common/include/global/sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
range	register REAL	The square of the two-dimensional distance

Table 2.5-224: compute_sqr_range Information.

2.5.10.10 tracks_stat.c
(/simnet/release/src/vehicle/libsrc/libbigwh/tracks_stat.c)

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"bigwh_loc.h"

2.5.10.10.1 bigwheel_left_track_broken

This routine is called to indicate that the left track has thrown.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_bigwheel	pointer to int	Standard

Table 2.5-225: bigwheel_left_track_broken Information.

2.5.10.10.2 bigwheel_right_track_broken

This routine is called to indicate that the right track has thrown.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_bigwheel	pointer to int	Standard

Table 2.5-226: bigwheel_right_track_broken Information.

2.5.10.10.3 bigwheel_repair_tracks

This routine is called to repair a thrown track.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_bigwheel	pointer to int	Standard

Table 2.5-227: bigwheel_repair_tracks Information.

2.5.10.11 veh_init.c
(/simnet/release/src/vehicle/libsrc/libbigwh/veh_init.c)

This file initializes space. It allocates space when the vehicle is initialized and deallocates when the vehicle is unitialized.

2.5.11 libterrain

(/simnet/release/src/libsrc/libterrain [libterrain])

As the database hands terrain polygons to the simulation, the bigwheel points must be checked for inclusion on the new polygons. The soil type of the new polygons must be available to the vehicle specific drivetrain simulation for drag computation, and the bounding volumes of any structures on the polygon must passed on for collision checking. This library provides all the routines for these computations, and communicates them mostly to libbigwheel.

This file preprocesses the terrain polygons which are received from the CIG. This processing allows for quick point inclusion and should work quickly, even on detailed micro-terrain.

Approximately every 37 ticks, a patch of local terrain is received from the graphics box. This patch contains every polygon and bounding volume that is within a 250 x 250 meter area centered around the vehicle's location rounded to the nearest 125 meters (in both x and y). The local world is divided up into four "buckets", which are conceptually the terrain to the southwest, southeast, northwest, and northeast of the rounded location.

The bucket (or buckets) that each polygon or bvol falls in is determined, and a pointer is placed to each polygon or bvol in the bucket. Then, when an elevation is calculated at a particular point, only one fourth as many polygons, on average, must be considered.

When **terrain_calc_elev()** is called to determine the elevation of a supporting polygon at a given (x,y) location, the following point inclusion algorithms are used to determine which polygons support that location:

1) Determine which bucket the (x,y) location falls in, and only look at the polygons in that bucket.
2) Check against the bounding box that surrounds the polygon. If the check fails, go on to the next polygon.
3) Perform a true point inclusion check, using the edges of the polygon. This algorithm is described for the routne **terrain_inside()**.

2.5.11.1 calc_elev.c
(/simnet/release/src/libsrc/libterrain/calc_elev.c)

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"terrain_loc.h"
"libmatrix.h"
"bbd.h"

Procedure Declarations:
terrain_inside()
check_polys_incl()
check_bvols_incl()
terrain_get_height()
terrain_make_edges()
terrain_make_normal()

Defines:

GET_CROSS() -- This macro is called to determine on which side of a line segment the given (x,y) point lies. The value returned is important only for its signum (i.e. whether it is positive or negative). A return of 0.0 indicates that the point lies directly on the line segment. The macro is called GET_CROSS because it computes a cross product of two vectors in the x-y plane. The result is the z-value of the resulting vector.

2.5.11.1.1 terrain_calc_elev

This routine is called to determine the height of supporting terrain at a given (x,y) point. If no supporting terrain is found, -1.0 is returned. Note that currently, the highest terrain at any given point is returned, if multiple supporting surfaces are found.

The parameter, *out_patch*, is a pointer to the chunk of local terrain information passed in by the graphics handler. The parameter, *location*, is the given (x, y) point at which to determine the height.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_patch	pointer to int	Standard
location	VECTOR	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i_morr	int	Standard
loc_patch	pointer to TERRAIN_PATCH	terrain_loc.h
support_poly_found	register int	Standard
local_x	register int	Standard
local_y	register int	Standard
cur_height	REAL	/simnet/common/include/global/sim_types.h
cur_bucket	pointer to register BUCKET	terrain_loc.h
Return Values		
Return Value	**Type**	**Meaning**
-1.0	REAL	no supporting terrain
cur_height	REAL	the height of the supporting terrain

Table 2.5-228: terrain_calc_elev Information.

2.5.11.1.2 check_polys_incl

This routine checks all polygons in the bucket, *cur_bucket*, against the location, *location*. A supporting polygon at the location is found through the following steps:
1) Check the polygon's bounding box polygon against the point. If the point falls outside the bounding box, then the point falls outside the polygon. Continue to the next polygon.
2) If the array of edges associated with the polygon has not been calculated yet, then call **terrain_make_edges**().
3) Use the array of edges to do a true point inclusion check, described in the routine **terrain_inside**(). If this check fails, then the point definitely falls outside the polygon. Continue to the next polygon. Otherwise, a height needs to be calculated.
4) If a normal to the polygon has not yet been created, then call **terrain_make_normal**() to create a normal, which is not necessarily a unit normal.
5) Call **terrain_get_height**() to determine the height at that location. If the returned height is higher than any height found so far, then it becomes the current height.
After all the polygons have been examined, return the flag, *support_poly_found*, if a support polygon was found. The highest height is pointed to by the variable *cur_height*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
cur_bucket	pointer to BUCKET	terrain_loc.h
location	pointer to register REAL	/simnet/common/include/global/sim_types.h
cur_height	pointer to REAL	/simnet/common/include/global/sim_types.h
soil_type	pointer to int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
cur_poly	pointer to register BBN_POLY_ENTRY	terrain_loc.h
incl_field	pointer to register BBN_INCL_INFO	terrain_loc.h
vertices	pointer to register R4P3D	/simnet/common/include/global/dgi_stdg.h
support_poly_found	int	Standard
test_height	register REAL	/simnet/common/include/global/sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
support_poly_found	int	a support polygon was found

Calls	
Function	**Where Described**
terrain_make_edges	Section 2.5.11.1.7
terrain_inside	Section 2.5.11.1.5
terrain_make_normal	Section 2.5.11.1.6
terrain_get_height	Section 2.5.11.1.4

Table 2.5-229: check_polys_incl Information.

2.5.11.1.3 check_bvols_incl

This routine checks all bvols in the given bucket, *cur_bucket*, against the given location, *location*. A supporting bvol at the location is found through the following steps. Note the similarities to the routine **check_polys_incl**():
1) Check the bounding volume against the point. It will be obvious if the point falls outside the bounding volume. Continue to the next bounding volume.
2) If the array of edges associated with the bounding volume has not been calculated yet, then call **terrain_make_edges**().
3) Use the array of edges to do a true point inclusion check, described in the routine **terrain_inside**(). If this check fails, then the point definitely falls outside the bvol. Continue to the next bvol. Otherwise, a height needs to be calculated.
4) Since bvols are flat, there is no point in creating a normal to the bvol. Instead, the height is calculated. Since bvols have a uniform height, there is no need to call **terrain_get_height**() to determine the height of the bvol. If the calculated height is higher than any height found so far, then it becomes the current height.
After all the polygons have been examined, return the flag, *support_bvol_found*, if a support bvol was found. The highest height is pointed to by the variable *height*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
cur_bucket	pointer to BUCKET	terrain_loc.h
location	pointer to register REAL	/simnet/common/include/global/sim_types.h
height	pointer to REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
incl_field	pointer to register BBN_INCL_INFO	terrain_loc.h
cur_bvol	pointer to register BBN_BVOL_ENTRY	terrain_loc.h
height_field	pointer to register BBN_HEIGHT_INFO	terrain_loc.h
test_height	register REAL	/simnet/common/include/global/sim_types.h
support_bvol_found	int	Standard
i	register int	Standard

Return Values		
Return Value	**Type**	**Meaning**
support_bvol_found	int	a supporting bounding volume was found

Calls	
Function	**Where Described**
terrain_make_edges	Section 2.5.11.1.7
terrain_inside	Section 2.5.11.1.5

Table 2.5-230: check_bvols_incl Information.

2.5.11.1.4 terrain_get_height

This routine is called by **check_polys_incl()** when the (x,y) point, *pt*, has passed the inclusion checks for the particular polygon. This routine returns the height of the point on the polygon at that location. The algorithm uses the fact that the dot product of a vector normal to the polygon with a vector that lies in the plane of the polygon must be equal to zero. The normal vector to the polygon will already have been determined. Since a vector which lies in the plane of the polygon can be determined by subracting any vertex of the polygon from the (x,y) point being checked, the dot product of these two vectors can be set to zero, leaving only one unknown, the z value of the vector that lies in the plane of the polygon. Solving for this value gives the height at the (x,y) location above the z value of the vertex used.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pt	pointer to register REAL	/simnet/common/include/global/sim_types.h
height_field	pointer to register BBN_HEIGHT_INFO	terrain_loc.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
vert	pointer to register REAL	/simnet/common/include/global/sim_types.h
norm	pointer to register REAL	/simnet/common/include/global/sim_types.h
height	register REAL	/simnet/common/include/global/sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
height	REAL	the height of the polygon at the (x,y) location

Table 2.5-231: terrain_get_height Information.

2.5.11.1.5 terrain_inside

This routine implements a true point inclusion algorithm by determining if the point being tested falls on the same side of every edge of the polygon. This algorithm assumes that all polygons are convex, and that a list of edges has already been made.

The cross product is taken (using only the x and y components) of a vector which represents an edge between two vertices of the polygon, and a vector which is drawn from the first vertex to the point being tested. Only the z value of the resulting vector is non-zero. To determine which side of an edge the point lies on, the signum of the z value of the resultant vector is examined. If the z value is zero, then the point lies on the edge, which neither allows nor prevents inclusion. If the point being tested falls on the same side of every edge of the polygon, then the signum of the z value will be the same for every edge. Therefore, the psuedo-code for this algorithm is:

```
signum = -1
do      {
        ret = check the z value of the cross porduct
        if ( ret != 0.0)
                signum = ( ret > 0.0 );
        increment vertex counter
        }
while ( signum != -1 )

while (  vertex counter != num_vertices )
        {
        ret = check the z value of the cross porduct
        if ( signum != ( ret > 0.0 ) && ret != 0.0 )
                return ( FALSE )
        increment vertex counter
        }
return ( TRUE )
```

The parameters to the routine represent the following:
pt -- the point being tested
incl_info -- provides the list of edges for the polygon
vertex_list -- the listing of vertex points for the polygon
num_verts -- the number of vertices in the polygon

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pt	pointer to REAL	/simnet/common/include/global/sim_types.h
incl_info	pointer to BBN_INCL_INFO	terrain_loc.h
vertex_list	pointer to R4P3D	/simnet/common/include/global/dgi_stdg.h
num_verts	int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
x	REAL	/simnet/common/include/global/sim_types.h
y	REAL	/simnet/common/include/global/sim_types.h
vert_cnt	int	Standard
dir	int	Standard
z_value	REAL	/simnet/common/include/global/sim_types.h
curr_edge	pointer to REAL	/simnet/common/include/global/sim_types.h
curr_vertex	pointer to R4P3D	/simnet/common/include/global/dgi_stdg.h

Return Values		
Return Value	**Type**	**Meaning**
INSIDE	int	the point is located inside the polygon
OUTSIDE	int	the point is located outside the polygon

Calls	
Function	**Where Described**
GET_CROSS	

Table 2.5-232: terrain_inside Information.

2.5.11.1.6 terrain_make_normal

This routine creates a normal to the polygon by taking the cross product of two of the vectors which represent two of its edges. The routine is also used to fill in portions of the BBN_HEIGHT_INFO structure from information passed from the graphics box. Note that this routine is only called for polygons, not bvols, since by definition, all bvols are flat.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vertex	pointer to R4P3D	/simnet/common/include/global/dgi_stdg.h
edge_list	pointer to VECTOR	/simnet/common/include/global/sim_types.h
height_field	pointer to BBN_HEIGHT_INFO	terrain_loc.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
first	pointer to register REAL	/simnet/common/include/global/sim_types.h
second	pointer to register REAL	/simnet/common/include/global/sim_types.h
normal	pointer to register REAL	/simnet/common/include/global/sim_types.h
init_pt	pointer to register REAL	/simnet/common/include/global/sim_types.h

Table 2.5-233: terrain_make_normal Information.

2.5.11.1.7 terrain_make_edges

This routine uses the vertices of a polygon to create a vector for each edge of the polygon.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vertices	pointer to R4P3D	/simnet/common/include/global/dgi_stdg.h
edge_list	pointer to REAL	/simnet/common/include/global/sim_types.h
vert_cnt	register int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
current	pointer to register R4P3D	/simnet/common/include/global/dgi_stdg.h
next	pointer to register R4P3D	/simnet/common/include/global/dgi_stdg.h
edge	pointer to register REAL	/simnet/common/include/global/sim_types.h

Table 2.5-234: terrain_make_edges Information.

2.5.11.2 get_size.c

(/simnet/release/src/libsrc/libterrain/get_size.c)

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"terrain_loc.h"

2.5.11.2.1 terrain_get_patch_size

This routine is used to determine the size of a specific patch of terrain, and is called by those who need to allocate their own patch.

Return Values		
Return Value	**Type**	**Meaning**
sizeof(TERRAIN_PATCH)	int	Size of TERRAIN_PATCH in bytes.

Table 2.5-235: terrain_get_patch_size Information.

2.5.11.3 get_soil.c
(/simnet/release/src/libsrc/libterrain/get_soil.c)

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"terrain_loc.h"

2.5.11.3.1 terrain_get_terrain_type

This routine is called by the tracks dynamics to determine what type of soil the tank is travelling.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_patch	pointer to int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
loc_patch	pointer to TERRAIN_PATCH	terrain_loc.h
Return Values		
Return Value	**Type**	**Meaning**
SOIL_ROAD	int	the current soil is of type SOIL_ROAD
loc_patch->cur_soil_type	int	the current soil type

Table 2.5-236: terrain_get_terrain_type Information.

2.5.11.4 lt_inited.c
(/simnet/release/src/libsrc/libterrain/lt_inited.c)

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"terrain_loc.h"

2.5.11.4.1 terrain_lt_inited

This routine is called by kinematics during the initial frames until the first terrain patch is received. If this returns FALSE, then kinematics will not initialize.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_patch	pointer to int	Standard
Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	the local terrain has not been initialized
((TERRAIN_PATCH *) out_patch) ->terrain_inited	int	the local terrain has been initialized

Table 2.5-237: terrain_lt_inited Information.

2.5.11.5 obstacles.c

(/simnet/release/src/libsrc/libterrain/obstacles.c)

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"terrain_loc.h"

2.5.11.5.1 terrain_obstructed

This routine is called to determine if the area at the given location with the given radius is obstructed by any bounding volume. If the area is not obstructed, 0 is returned. Otherwise, the type of the obstructing bvol is returned.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_patch	pointer to int	Standard
location	pointer to REAL	/simnet/common/include/global/sim_types.h
radius	REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
local_x	register int	Standard
local_y	register int	Standard
cur_bucket	pointer to register BUCKET	terrain_loc.h
cur_bvol	pointer to register BBN_BVOL_ENTRY	terrain_loc.h
minx	short	Standard
miny	short	Standard
maxx	short	Standard
maxy	short	Standard
loc_patch	pointer to TERRAIN_PATCH	terrain_loc.h
Return Values		
Return Value	**Type**	**Meaning**
-1.0	unsigned short	the procedure failed
FALSE	unsigned short	the location is not obstructed by a bounding volume
cur_bvol->bvol.type_id	unsigned short	the location is obstructed by a bounding volume

Table 2.5-238: terrain_obstructed Information.

2.5.11.6 preproc.c
(/simnet/release/src/libsrc/libterrain/preproc.c)

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"terrain_loc.h"
"bbd.h"
"libmatrix.h"

Declarations:
terrain_add_poly_ptr()
terrain_add_bvol_ptr()
dgi_verbose

2.5.11.6.1 terrain_preproc_terrain

This is the driver routine for setting up the preprocessed polygons and bvols. First, it initializes the array of buckets to zeros. Then, for each polygon and bvol it 1) sets the *edges_made* and *normal_made* flags to FALSE, 2) bcopies the polygon into local structures, and 3) calls **add_poly_ptr**() to place a pointer to the polygon in each bucket that the polygon overlaps.

The parameters represent the following:
out_patch -- local terrain information received from the graphics box
num_polys -- the number of polygons
dgi_poly_array[] --the array of polygons in the Delta Graphics, Inc. format
num_bvols -- the number of bounding volumes
dgi_bvol_array[] -- the array of bvols in the Delta Graphics, Inc. format
position -- the (x,y,z) coordinates of the current position.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
out_patch	pointer to int	Standard
num_polys	register int	Standard
dgi_poly_array[]	LT_POLY_ENTRY	/simnet/common/include/cig_if/sim_cig_if.h
num_bvols	register int	Standard
dgi_bvol_array[]	LT_BVOL_ENTRY	/simnet/common/include/cig_if/sim_cig_if.h
position	pointer to REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
loc_patch	pointer to register TERRAIN_PATCH	terrain_loc.h
i	register int	Standard
j	int	Standard
dgi_poly	pointer to register LT_POLY_ENTRY	/simnet/common/include/cig_if/sim_cig_if.h
(dgi_bvol	pointer to register LT_BVOL_ENTRY	/simnet/common/include/cig_if/sim_cig_if.h
loc_poly	pointer to register BBN_POLY_ENTRY	terrain_loc.h
loc_bvol	pointer to register BBN_BVOL_ENTRY	terrain_loc.h

Errors	
Error	**Reason for Error**
stderr	PANIC standard bogosities located in preproc_terrain

Calls	
Function	**Where Described**
terrain_add_poly_ptr	Section 2.5.11.6.2
terrain_add_bvol_ptr	Section 2.5.11.6.3

Table 2.5-239: terrain_preproc_terrain Information.

2.5.11.6.2 terrain_add_poly_ptr

This routine takes a polygon, *poly_to_add*, and places a pointer to it in the bucket array. Generally, all terrain polygons will be on a 125 meter grid, relative to the middle x and middle y for the patch of terrain. However, since the terrain polygons became relaxed, it is now possible that a polygon may cover more than one bucket. Additionally, bvols, object polygons, and terrain polygons associated with micro terrain are not constrained to fall on regular grid at all, and may also overlap more than one bucket.

To calculate which bucket(s) the polygon belongs in, the bounding box of the polygon is examined. Note that when the coordinates that define the bounding box are accessed, they are saved for future use by the first inclusion algorithm. Then, a pointer to the polygon is placed in every bucket that the polygon's bounding box overlaps.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
loc_patch	pointer to TERRAIN_PATCH	terrain_loc.h
poly_to_add	pointer to register BBN_POLY_ENTRY	terrain_loc.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
terr_ptr	pointer to register BUCKET	terrain_loc.h
cur_poly	pointer to register LT_POLY_ENTRY	/simnet/common/include/cig_if/sim_cig_if.h
incl_field	pointer to register BBN_INCL_INFO	terrain_loc.h
i	register int	Standard
j	register int	Standard
min_x	register int	Standard
min_y	register int	Standard
max_x	register int	Standard
max_y	register int	Standard

Table 2.5-240: terrain_add_poly_ptr Information.

2.5.11.6.3 terrain_add_bvol_ptr

This routine places the passed bvol pointer, *bvol_to_add*, into the bucket array of the local terrain patch. Bvols are not constrained to fall on regular grid and may overlap more than one bucket.

To calculate which bucket(s) the bvol belongs in, the bounding box of the bvol is examined. Note that when the coordinates that define the bounding box are accessed, they are saved for future use by the first inclusion algorithm. Then, a pointer to the bvol is placed in every bucket that the bvol's bounding box overlaps.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
loc_patch	pointer to TERRAIN_PATCH	terrain_loc.h
bvol_to_add	pointer to register BBN_BVOL_ENTRY	terrain_loc.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
terr_ptr	pointer to register BUCKET	terrain_loc.h
cur_bvol	pointer to register LT_BVOL_ENTRY	/simnet/common/include/cig_if/sim_cig_if.h
incl_field	pointer to register BBN_INCL_INFO	/simnet/common/include/cig_if/sim_cig_if.h
i	register int	Standard
j	register int	Standard
min_x	register int	Standard
min_y	register int	Standard
max_x	register int	Standard
max_y	register int	Standard

Table 2.5-241: Information.

2.5.11.7 terr_init.c
(/simnet/release/src/libsrc/libterrain/terr_init.c)

2.5.11.8 terrain_loc.h
(/simnet/release/src/libsrc/libterrain/terrain_loc.h)

Include:

"sim_types.h"
"sim_dfns.h"
"sim_macros.h"

Defines:

TERRAIN_DEBUG
MAX_POLYS_BVOLS_PATCH
MAX_POLYS_BVOLS_LM

TREE_PRI
STATIC_MODEL_PRI

TERRAIN_PRI
OVERLAY_PRI
OVER_OVERLAY_PRI
MOVING_MODEL_PRI
MODEL_OVERLAY_PRI
UNCOND_PRI

INSIDE
OUTSIDE

Typedefs:

BBN_INCL_INFO	--	contains all the information needed for the inclusion algorithms
BBN_HEIGHT_INFO	--	contains all the information needed for the height determination algorithms
BBN_POLY_ENTRY		
BBN_BVOL_ENTRY		
BUCKET		
TERRAIN_PATCH		

2.5.11.9 verb_mode.c

(/simnet/release/src/libsrc/libterrain/verb_mode.c)

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"terrain_loc.h"

2.5.11.9.1 terrain_verbose_mode_on

This routine is called by m1_main when the "-v" flag is given as an option.

2.5.12 librva

(./simnet/release/src/libsrc/librva [librva])

The librva CSU maintains, on a tick-by-tick basis, an accurate list of vehicles in the simulated world. The librva processes Vehicle Appearance Packets (VAP) from other vehicles on the SIMNET network. When a VAP is received, librva places the updated information into its Remote Vehicle Approximation (RVA) table. The RVA table reflects the changes in vehicle appearances. Once each frame, each vehicle is dead reckoned, vehicles not heard from in the last 12 seconds are timed out, and vehicles in close proximity are checked for servicing needs and collisions. For efficiency, the librva only updates its tables of vehicle appearance when the vehicle's appearance has varied in certain ways.

The libmsg module is informed of newly arrived vehicles on the network and vehicles that have disappeared, thereby keeping the CIG informed of the vehicles it must display.

The librva module maintains priority lists of the simulated vehicles according to their vehicle type, force alignment, and range. When a Vehicle Appearance Packet is placed in the RVA table, it is also linked into one of several vehicle lists, each of which represents a different priority level. The highest priority N vehicles are actually processed, while any other Vehicle Appearance Packets are ignored. N is bound at run-time, and is typically 64. In most cases, fewer than 64 vehicles are ever within visual range of the simulated vehicle.

2.5.12.1 adj_veh_app.c

(./simnet/release/src/libsrc/librva/adj_veh_app.c)

This file provides the capabilities of changing the appearance of remote vehicles.

Includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"libveh.h"
"pro_sim.h"
"rva_loc.h".

2.5.12.1.1 rva_adjust_veh_appear

This routine tells the CIG to change the appearance of the remote vehicle, *veh_id*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
guises	pointer to VehicleGuises	/simnet/common/include/protocol/basic.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
r	pointer to register RVA_ENTRY	librva.h
hash_id	int	Standard

Errors	
Error	**Reason for Error**
PRINT_VID_ERROR	Cannot adjust appearance of invalid veh_id

Calls	
Function	**Where Described**
rva_find_hash_entry	Section 2.5.12.11.11
map_net_to_cig	Section 2.6.11.5.8

Table 2.5-242: rva_adust_veh_appear Information.

2.5.12.1.2 rva_reset_veh_appear

This routine tells the CIG to reset the remote vehicle specified by *veh_id* to its original appearance.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
r	pointer to RVA_ENTRY	librva.h
hash_id	int	Standard

Errors	
Error	**Reason for Error**
PRINT_VID_ERROR	Cannot reset appearance of invalid veh_id

Calls	
Function	**Where Described**
rva_find_hash_entry	Section 2.5.12.11.11
map_net_to_cig	Section 2.6.11.5.8

Table 2.5-243: rva_reset_veh_appear Information.

2.5.12.2 debug.c

(./simnet/release/src/libsrc/librva/debug.c)

This file is used for debugging purposes.

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"pro_sim.h"
"rva_loc.h"
"prior_loc.h".

2.5.12.2.1 rva_turn_debug_on

This procedure sets the value of *rva_debug* to TRUE.

2.5.12.2.2 rva_turn_debug_off

This procedure sets the value of *rva_debug* to FALSE.

2.5.12.2.3 rva_dump_priority_lists

This procedure prints the priority list (for debugging purposes).

2.5.12.3 forget_veh.c
(./simnet/release/src/libsrc/librva/forget_veh.c)

This file includes:
- "stdio.h"
- "math.h"
- "sim_dfns.h"
- "sim_macros.h"
- "sim_types.h"
- "mass_stdc.h"
- "dgi_stg.h"
- "sim_cig_if.h"
- "pro_sim.h"
- "libkin.h"
- "bigwheel.h"
- "rva_loc.h"
- "prior_loc.h"
- "libhull.h".

2.5.12.3.1 rva_forget_about_vehicle

This routine removes a vehicle specified by *veh_id* from the RVA table. If the static vehicle has the "is_static" flag set to TRUE, then the vehicle must also removed from the CIG using "tell_cig.c". Otherwise, the CIG never heard about it. The vehicle will be deleted from the priority lists the next time the priority lists are adjusted.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
r	register pointer to RVA_ENTRY	librva.h
hash_id	int	Standard

Calls	
Function	**Where Described**
rva_find_hash_entry	Section 2.5.12.11.11
collision_forget_about	Section 2.5.10.6.4
prior_debug	Macro defined in Section 2.5.12.17 prior_loc.h
delete_veh_from_cig_msg	Section 2.1.2.2.2.27.1

Called By	
Function	**Where Described**
delete_or_timeout	Section 2.5.12.15.3
adjust_dynamic_vehicles	Section 2.5.12.15.5
adjust_static_vehicles	Section 2.5.12.15.6
process_known_dynamic	Section 2.5.12.20.3

Table 2.5-244: rva_forget_about_vehicle Information.

2.5.12.3.2 delete_vehicles_from_list

This routine deletes every vehicle from the specified priority list, *pri_list*.. As it deletes a vehicle, it also asks the CIG interface code to delete the OTHERVEH_STATE copy of the vehicle, if that copy exists.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pri_list	pointer to PRIORITY_LIST	prior_loc.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
veh	pointer to RVA_ENTRY	librva.h

Errors	
Error	**Reason for Error**
PRINT_VID_ERROR	Invalid vehicle to delete_from_lists()
Calls	
Function	**Where Described**
collision_forget_about	Section 2.5.10.6.4
prior_degug	Macro defined in Section 2.5.12.17
delete_veh_from_cig_msg	Section 2.1.2.2.2.27.1

Table 2.5-245: delete_vehicles_from_list Information.

2.5.12.4 get_air_vehs.c

(./simnet/release/src/libsrc/librva/get_air_vehs.c)

This file includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"pro_sim.h"
"rva_loc.h".

2.5.12.4.1 rva_get_air_veh_list

This routine maintains a list of air vehicles and indirectly returns values for the air vehicle list and number of air vehicles on the list through the *veh_list* and the *num_vehs* pointers.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_list	pointer to pointer to array of RVA_ENTRY	librva.h
num_vehs	pointer to int	Standard

Table 2.5-246: rva_get_veh_list Information.

2.5.12.5 get_obj_type.c
(./simnet/release/src/libsrc/librva/get_obj_type.c)

This file includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"pro_sim.h"
"rva_loc.h".

2.5.12.5.1 rva_get_object_type

Given a *hash_id*, this routine returns the object type of the RVA Table entry.

Parameters		
Parameter	Type	Where Typedef Declared
hash_id	int	Standard
Return Values		
Return Value	Type	Meaning
remote_vehicles[hash_id]. rva_entry.pkt.guises. distinguished	ObjectType	The object type of the RVA entry for the distinguished force ID.
remote_vehicles[hash_id]. rva_entry.pkt.guises. other	ObjectType	The object type of the RVA entry for otherForceID

Table 2.5-247: rva_get_object_type Information.

2.5.12.6 get_prior_list.c
(./simnet/release/src/libsrc/librva/get_pri_list.c)

This file includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"simstdio.h"
"rva_loc.h"
"prior_loc.h"
"libhull.h"
"libkin.h"
"libveh.h".

2.5.12.6.1 rva_get_priority_list

This routine returns the index of the priority list given a specific vehicle appearance variant, *vap*, and distance, *r_squared*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vap	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
r_squared	double	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
curr_list	pointer to PRIORITY_LIST	Section 2.5.12.17, prior_loc.h
hash_id	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
i	int	the priority_list index for the specified vap
-1	int	out of visual range, therefore, no priority list was found for the vehicle

Calls	
Function	**Where Described**
is_friendly	Section 2.6.10.6.1

Table 2.5-248: rva_get_priority_list Information.

2.5.12.7 get_vap.c
(./simnet/release/src/libsrc/librva/get_vap.c)

This file includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"pro_sim.h"
"rva_loc.h"

2.5.12.7.1 rva_get_veh_app_pkt

This routine returns a pointer to the Vehicle Appearance Variant, *vap*, of the Vehicle ID specified in *hash_id*.

Parameters		
Parameter	Type	Where Typedef Declared
hash_id	int	Standard
Internal Variables		
Variable	Type	Where Typedef Declared
r	register pointer to RVA_ENTRY	librva.h
Return Values		
Return Value	Type	Meaning
&r -> pkt	pointer to VehicleAppearanceVariant	a pointer to the vehicle appearance packet of the specified vehicle id.
NULL	pointer to VehicleAppearanceVariant	either the hash_id is equal to hash_IDIrrelevant or there is no packet.

Table 2.5-249: rva_get_veh_app_pkt Information.

2.5.12.7.2 rva_get_rva_entry

This routine returns a pointer to the RVA Table entry for the Vehicle ID specified in *hash_id*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
hash_id	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
r	register pointer to RVA_ENTRY	librva.h
Return Values		
Return Value	**Type**	**Meaning**
r	pointer to RVA_ENTRY	a pointer to the remote_vehicles rva entry
NULL	pointer to RVA_ENTRY	there is no remote_vehicles rva entry

Table 2.5-250: rva_get_rva_entry Information.

2.5.12.8 get_veh_loc.c
(./simnet/release/src/libsrc/librva/get_veh_loc.c)

This file includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"pro_sim.h"
"rva_loc.h"

2.5.12.8.1 rva_get_veh_loc

This routine returns the remote vehicles location of the Vehicle ID specified in *hash_id*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
hash_id	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
remote_vehicles[hash_id].rva_entry.pkt.location	pointer to double	a pointer to the remote vehicles location

Table 2.5-251: rva_get_veh_loc Information.

2.5.12.9 get_vehs.c
(./simnet/release/src/libsrc/librva/get_vehs.c)

This file includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"pro_sim.h"
"rva_loc.h"
"prior_loc.h"

2.5.12.9.1 rva_get_close_list

The librva maintains a close vehicles list. Close vehicles are defined as those vehicles within 3500 meters. This routine sets the *veh_list* pointer equal to the close vehicle list (*close_veh_list*) and sets the *num_vehs* pointer equal to the number of close vehicles (*num_close_vehs*).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_list	pointer to pointer to array of RVA_ENTRY	librva.h
num_vehs	pointer to int	Standard

Table 2.5-252: rva_get_close_list Information.

2.5.12.9.2 rva_get_lists

This routine gets various priority lists:

1) The *static_list* pointer is set to the static vehicles list (*static_veh_list*) and the *num_static* pointer is set to the number of static vehicles (*num_static_vehs*).

2) The *rm_list* pointer is set to the remove vehicles list (*remove_veh_list*) and the *num_rm* pointer is set to the number of vehicles to remove (*num_to_remove*).

3) The *chg_list* pointer is set to the change vehicles list (*chg_veh_list*) and the *num_chg* pointer is set to the number of vehicles to change (*num_chg_vehs*).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
static_list	pointer to pointer to array of RVA_ENTRY	librva.h
num_static	pointer to int	Standard
rm_list	pointer to pointer to array of RVA_ENTRY	librva.h
num_rm	pointer to int	Standard
chg_list	pointer to pointer to RVA_ENTRY	librva.h
num_chg	pointer to int	Standard

Table 2.5-253: rva_get_lists Information.

2.5.12.9.3 rva_get_num_hash_entries

This routine returns the number of hash entries.

Return Values		
Return Value	**Type**	**Meaning**
N_HASH_ENTRIES	int	The number of hash entries

Table 2.5-254: rva_get_num_hash_entries Information.

2.5.12.9.4 rva_get_num_close_vehs

This routine returns the number of close vehicles.

Return Values		
Return Value	**Type**	**Meaning**
num_close_vehs	int	The number of close vehicles

Table 2.5-255: rva_get_num_close_vehs Information.

2.5.12.9.5 rva_get_num_air_vehs

This routine returns the number of air vehicles.

Return Values		
Return Value	**Type**	**Meaning**
num_air_vehs	int	The number of air vehicles

Table 2.5-256: rva_get_num_air_vehs Information.

2.5.12.9.6 set_save_num_static_vehs

This routine sets the saved number of static vehicles in *save_num_static_vehs* to *n*.

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

Table 2.5-257: set_save_num_static_vehs Information.

2.5.12.9.7 rva_get_num_static_vehs

This routine returns the number of saved static vehicles

Return Values		
Return Value	**Type**	**Meaning**
save_num_static_vehs	int	The number of saved static vehicles

Table 2.5-258: rva_get_num_static_vehs Information.

2.5.12.9.8 rva_get_num_mvg_vehs

This routine returns the number of saved moving vehicles.

Return Values		
Return Value	**Type**	**Meaning**
save_mum_mvg_vehs	int	The number of saved moving vehicles

Table 2.5-259: rva_get_num_mvg_vehs Information.

2.5.12.10 get_vid.c
(./simnet/release/src/libsrc/librva/get_vid.c)

This file includes:
- "stdio.h"
- "math.h"
- "sim_dfns.h"
- "sim_macros.h"
- "sim_types.h"
- "mass_stdc.h"
- "dgi_stg.h"
- "sim_cig_if.h"
- "pro_sim.h"
- "rva_loc.h"

2.5.12.10.1 rva_get_veh_id

Given the argument *hash_id*, this routine returns a pointer to the associated remote vehicles RVA_ENTRY Vehicle ID.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
hash_id	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
r	register pointer to RVA_ENTRY	librva.h
Return Values		
Return Value	**Type**	**Meaning**
&r -> pkt.vehicleID	pointer to VehicleID	a pointer to the vehicle ID of the specified hash id.
NULL	pointer to VehicleID	either the hash_id is equal to hash_IDIrrelevant or there is no VehicleID entry.

Table 2.5-260: rva_get_veh_id Information.

2.5.12.11 hash.c
(./simnet/release/src/libsrc/librva/hash.c)

This file includes:
- "stdio.h"
- "sim_dfns.h"
- "sim_macros.h"
- "sim_types.h"
- "mass_stdc.h"
- "dgi_stg.h"
- "sim_cig_if.h"
- "assoc.h"
- "rva_loc.h"
- "prior_loc.h"

The following procedures are declared as static:
- **find_hash_value()**
- **free_hash_entry()**
- **get_hash_entry()**

The following variables are declares:
- remote_vehicles
- rva_hash_table

2.5.12.11.1 rva_alloc_hash_table

This routine creates a hash table of the specified size. The *table* pointer represents the hash table, with the number of entries equal to *n_entries*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
table	pointer to pointer to HASH_TABLE	Section 2.5.12.24, rva_loc.h
n_entries	int	Standard

Return Values		
Return Value	**Type**	**Meaning**
-1	int	The routine failed
0	int	The table was successfully created

Table 2.5-261: rva_alloc_hash_table Information.

2.5.12.11.2 rva_init_hash_table

This routine initializes all the internal structures of the hash table, *table*, to hashIDIrrelevant, and establishes the free list of entries.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
table	pointer to HASH_TABLE	Section 2.5.12.24, rva_loc.h
n_entries	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard

Table 2.5-262: rva_init_hash_table Information.

2.5.12.11.3 rva_lookup_hash_table_entry

This routine returns *hashi*, the index into the given hash table, *table*, which contains the entry corresponding to the *vid* parameter, or hashIDIrrelevant if the entry does not exist.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
table	pointer to HASH_TABLE	Section 2.5.12.24, rva_loc.h
vid	register pointer to VehicleID	/simnet/common/include/protocol/basic.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
entry	register pointer to HASH_ENTRY	rva_loc.h
hashi	register int	Standard
Return Values		
Return Value	**Type**	**Meaning**
hashi	register int	The index into the given hash table which contains the entry corresponding to *vid*

Calls	
Function	**Where Described**
VEHICLE_IDS_EQUAL	Macro defined in /simnet/common/include/global/sim_macros.h

Table 2.5-263: rva_lookup_hash_table_entry Information.

2.5.12.11.4 rva_remove_hash_table_entry

This routine finds the hash value specified by *vid* in the hash table specified by *table*, removes the entry for its hash value (specified by *hashi*) from the list of entries, and pushes it onto the free list stack. It returns -1 if the entry does not exist and 0 otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
table	pointer to HASH_TABLE	Section 2.5.12.24, rva_loc.h
vid	register pointer to VehicleID	/simnet/common/include/protocol/basic.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
entry	register pointer to HASH_ENTRY	Section 2.5.12.24, rva_loc.h
prev_entry	register pointer to HASH_ENTRY	Section 2.5.12.24, rva_loc.h
hashi	register int	Standard
hash_value	register int	Standard

Return Values		
Return Value	**Type**	**Meaning**
0	int	The entry existed and was removed
hashIDirrelevant	int	if hashi is equal to hashIDirrelevant (which equals -1), the entry does not exist

Calls	
Function	**Where Described**
find_hash_value	Section 2.5.12.11.6
VEHICLE_IDS_EQUAL	Macro defined in /simnet/common/include/global/sim_macros.h
free_hash_entry	Section 2.5.12.11.7

Table 2.5-264: rva_remove_hash_table_entry Information.

2.5.12.11.5 rva_insert_hash_table_entry

This routine returns the index of a new hash table entry if one is available from the free list. Otherwise, hashIDIrrelevant is returned.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
table	pointer to HASH_TABLE	Section 2.5.12.24, rva_loc.h
vid	register pointer to VehicleID	/simnet/common/include/protocol/basic.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
hashi	register int	Standard
hash_value	register int	Standard
Return Values		
Return Value	**Type**	**Meaning**
hashi	register int	The index of the new entry
hashIDIrrelevant	int	if hashi is equal to hashIDIrrelevant (which equals -1), there are no more entries

Calls	
Function	**Where Described**
get_hash_entry	Section 2.5.12.11.8
find_hash_value	Section 2.5.12.11.6

Table 2.5-265: rva_insert_hash_table_entry Information.

2.5.12.11.6 find_hash_value

This routine returns the *hash_value* of the Vehicle specified in the *vid* argument.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vid	register pointer to VehicleID	/simnet/common/include/protocol/basic.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
hash_value	register long int	Standard
Return Values		
Return Value	**Type**	**Meaning**
hash_value	register long int	The hash value of the vid.

Called By	
Function	**Where Described**
rva_remove_hash_table_entry	Section 2.5.12.11.4
rva_insert_hash_table_entry	Section 2.5.12.11.5

Table 2.5-266: find_hash_value Information.

2.5.12.11.7 free_hash_entry

This routine frees the specified hash entry *(hashi)* of the specified hash table *(table)*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
table	register pointer to HASH_TABLE	Section 2.5.12.24, rva_loc.h
hashi	register pointer to int	Standard

Table 2.5-267: free_hash_entry Information.

2.5.12.11.8 get_hash_entry

This routine returns the next available hash entry in the free list for the specified hash table *(table)*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
table	register pointer to HASH_TABLE	Section 2.5.12.24, rva_loc.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
hashi	register int	Standard
Return Values		
Return Value	**Type**	**Meaning**
hashIDirrelevant	int	The entry does not exist
hashi	register int	The index of the entry

Table 2.5-268: get_hash_entry Information.

2.5.12.11.9 rva_alloc_rva_table

This routine allocates memory for the RVA hash table and sets remote_vehicles to point to the entries in the table.

Calls	
Function	**Where Described**
rva_alloc_hash_table	Section 2.5.12.11.1

Table 2.5-269: rva_alloc_rva_table Information.

2.5.12.11.10 rva_init_rva_table

This routine initializes the RVA hash table.

Calls	
Function	**Where Described**
rva_init_hash_table	Section 2.5.12.11.2

Table 2.5-270: rva_init_rva_table Information.

2.5.12.11.11 rva_find_hash_entry

This routine looks for the vehicle ID specified by *vid* in the RVA hash table.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vid	register pointer to VehicleID	/simnet/common/include/protocol/basic.h
Return Values		
Return Value	**Type**	**Meaning**
rva_lookup_hash_table_entry (rva_hash_table, vid)	int	The rva_hash_table entry.

Table 2.5-271: rva_find_hash_entry Information.

2.5.12.11.12 rva_delete_hash_entry

This routine removes the vehicle specified by *vid* from the RVA hash table.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vid	register pointer to VehicleID	/simnet/common/include/protocol/basic.h
Return Values		
Return Value	**Type**	**Meaning**
rva_remove_hash_table_entry(rva_hash_table, vid)	int	Either the entry existed and was removed, or the entry did not exist.

Table 2.5-272: rva_delete_hash_entry Information.

2.5.12.11.13 rva_add_hash_entry

This routine adds the vehicle specified by *vid* to the RVA hash table.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vid	register pointer to VehicleID	/simnet/common/include/protocol/basic.h
Return Values		
Return Value	**Type**	**Meaning**
rva_insert_hash_table_entry (rva_hash_table, vid)	int	Either the index of the new entry is returned, or there are no more entries.

Table 2.5-273: rva_add_hash_entry Information.

2.5.12.12 lock_veh.c

(./simnet/release/src/libsrc/librva/lock_veh.c)

This file includes:

"stdio.h"
"sim_dfns.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"rva_loc.h"
"prior_loc.h"
"libkin.h"
"libhull.h"

2.5.12.12.1 rva_lock_veh_into_buf

This routine puts the vehicle into the special priority list zero, which tells librva to smooth the location orientation between updates. The only way that it can be removed from list 0 is by moving out of range and timing out, or by a call to **rva_unlock_veh()**.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
r	pointer to RVA_ENTRY	librva.h
hash_id	int	Standard

Errors	
Error	**Reason for Error**
PRINT_VID_ERROR	vehicle is unknown
Calls	
Function	**Where Described**
rva_find_hash_entry	Section 2.5.12.11.11
zero_init_veh	Section 2.5.12.29.1

Table 2.5-274: rva_lock_veh_into_buf Information.

2.5.12.12.2 rva_unlock_veh

This routine removes a vehicle from priority list zero. It moves the vehicle back to its normal priority list.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
r	pointer to RVA_ENTRY	librva.h
list_num	int	Standard
hash_id	int	Standard
r_squared	double	Standard

Errors	
Error	**Reason for Error**
PRINT_VID_ERROR	vehicle is unknown
PRINT_VID_ERROR	vehicle is not on priority list zero

Calls	
Function	**Where Described**
rva_find_hash_entry	Section 2.5.12.11.1
zero_uninit_veh	Section 2.5.12.29.2
try_to_remove_veh	Section 2.5.12.18.1
kinematics_range_squared	Section 2.5.8.10.1
rva_get_priority_list	Section 2.5.12.6.1

Table 2.5-275: rva_unlock_veh Information.

2.5.12.13 markers.c
(./simnet/release/src/libsrc/librva/markers.c)

This file includes:
- "stdio.h"
- "math.h"
- "sim_dfns.h"
- "sim_macros.h"
- "sim_types.h"
- "mass_stdc.h"
- "dgi_stg.h"
- "sim_cig_if.h"
- "pro_sim.h"
- "pro_timers.h"
- "assoc.h"
- "timers.h"
- "timers_dfn.h"
- "libveh.h"
- "libmap.h"
- "libhull.h"
- "libkin.h"
- "rva_loc.h"
- "prior_loc.h"

Constant define: N_MARKER_ENTRIES

Variable and Procedure Declarations:
- marker_hash_tables
- **process_unknown_marker()**

Minefield markers are set up as static vehicles, which the CIG is able to process. Markers are included on the priority lists of static vehicles.

2.5.12.13.1 rva_alloc_marker_table

This routine allocates memory for markers.

Calls	
Function	**Where Described**
rva_alloc_hash_table	Section 2.5.12.11.1

Table 2.5-276: rva_alloc_marker_table Information.

2.5.12.13.2 rva_init_marker_table

This routine initializes the marker hash table.

Calls	
Function	**Where Described**
rva_init_hash_table	Section 2.5.12.11.2

Table 2.5-277: rva_init_marker_table Information.

2.5.12.13.3 rva_process_markers

This routine processes a Vehicle Appearance Packet for the minefield marker specified in *markers*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
markers	pointer to MarkerVariant	/simnet/common/include/protocol/basic.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
m	register pointer to MarkerDescriptor	/simnet/common/include/protocol/p_sim.h
id	VehicleID	/simnet/common/include/protocol/basic.h
hash_id	int	Standard
r	pointer to RVA_ENTRY	librva.h
Calls		
Function	**Where Described**	
rva_lookup_hash_table_entry	Section 2.5.12.11.3	
rva_process_unknown_marker	Section 2.5.12.13.4	

Table 2.5-278: rva_process_markers Information.

2.5.12.13.4 rva_process_unknown_marker

This routine creates a Vehicle Appearance Packet for the minefield marker specified in *marker*, which is not currently in the RVA table.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
marker	pointer to MarkerDescriptor	/simnet/common/include/protocol/p_sim.h
id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
guises	pointer to VehicleGuises	/simnet/common/include/protocol/basic.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
hash_id	int	Standard
r	pointer to RVA_ENTRY	librva.h
marker_to_world	T_MATRIX	/simnet/common/include/global/sim_types.h
theta	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
rva_insert_hash_table_entry	Section 2.5.12.11.5
timers_get_current_tick	Section 2.6.3.1.1
mat_rot_init	Section 2.6.2.47.1
d2f_mat_copy	Section 2.6.2.1.1

Table 2.5-279: rva_process_unknown_marker Information.

2.5.12.13.5 adjust_markers

This routine does the tick by tick adjustments on minefield markers. For every marker on the static vehicles list, the routine checks to see if the marker needs to be added, deleted, changed, or removed from the list. It also checks for timeouts.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
entry	register pointer to HASH_ENTRY	Section 2.5.12.24, rva_loc.h
tmp_veh	register pointer to RVA_ENTRY	librva.h
i	register int	Standard
curr_minus_timeout	int	Standard

Calls	
Function	**Where Described**
timers_get_current_tick	Section 2.6.3.1.1
rva_remove_hash_table_entry	Section 2.5.12.11.4

Table 2.5-280: adjust_markers Information.

2.5.12.14 prior_init.c
(./simnet/release/src/libsrc/librva/prior_init.c)

This file includes:
"stdio.h"
"math.h"
"simstdio.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"obj_type.h"
"cig_buffer.h"
"rva_loc.h"
"prior_loc.h"

This file defines the following constants:
numNameMaps
NUM_ALIGNMENTS

This file defines the following macros:
vehEnvironMask
vehClassMask
gndVehicle
airVehicle
error_exit
get_non_comment

This file declares:
obj_name_map[numNameMaps]
veh_alignments[NUM_ALIGNMENTS]
obj_classes[MAX_FILTER_CLASSES]

2.5.12.14.1 rva_priority_setup

This routine sets up the priority list. The priority classes are set up which determine which vehicles will be placed in which priority lists. Memory space is allocated for the priority lists, then the priority lists are initialized (including reserved priority list 0). This routine reads the object type, alignment (either "aligned both", "aligned foes", or "aligned friend"), and the minimum and maximum ranges. Note that a maximum range of -1 signifies using the maximum CIG range. The routine checks whether the vehicles on the priority list are allowed in the CIG buffer (the default allows the vehicles in the CIG buffer for backward compatibility). This routine communicates with libfilter to determine the filter classes.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pri_data_file	pointer to char	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
pri_fp	pointer to FILE	
i	int	Standard
j	int	Standard
ret	pointer to char	Standard
num_read	int	Standard
num_lists	int	Standard
curr_list	pointer to PRIORITY_LIST	Section 2.5.12.17, prior_loc.h
size	int	Standard
buf	array 80 of char	Standard
word	array 30 of char	Standard
add_dynamic_veh_list	pointer to pointer to RVA_ENTRY	librva.h
obj_name	array 50 of char	Standard
obj_num	int	Standard
min_dist	int	Standard
max_dist	int	Standard
allowed_in_buf	int	Standard

Errors	
Error	**Reason for Error**
error_exit	- can't open file: pri_data_file - unexpected EOF or blank line getting max range - unexpected EOF getting num_lists - unexpected blank line getting num_lists - unexpected EOL or EOF getting numbers of vehicles - can't malloc add_dynamic_eh_list - unexpected NULL line in list - bad vehicle name - unexpected blank line reading alignment - invalid alignment reading list - unexpected range entries for list - bad max_distance reading list - bad min_distance reading list - unexpected blank line reading filter class - invalid filter class reading list

Calls	
Function	**Where Described**
filter_set_max_cig_range	Section 2.5.14.8.2
get_non_comment	Macro defined in this file.
filter_add_class	Section 2.5.15.1.1
FCLOSE	/simnet/common/include/global/simstdio.h
filter_verify_classes	Section 2.5.14 libfilter

Table 2.5-281: rva_priority_setup Information.

2.5.12.15 prior_lists.c

(./simnet/release/src/libsrc/librva/prior_lists.c)

This file includes:

"stdio.h"
"math.h"
"sines.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"basic.h"
"obj_type.h"
"pro_timers.h"
"simstdio.h"
"libhull.h"
"timers_dfn.h"
"timers.h"
"libkin.h"
"bigwheel.h"
"rva_loc.h"
"prior_loc.h"

This file defines the following macros and constants:

COLLISION_RADIUS_SQUARED
SERVICE_RADIUS_SQUARED
FULL_CIRCLE
TICKS_PER_MINUTE
RPM_TO_CIRCLE_PER_TICK

2.5.12.15.1 check_very_close_veh

If a vehicle is very close (within 100 meters), this macro does an exact range calulation and checks to see if the vehicle is within range to be a service vehicle. If the vehicle is even closer, then the vehicle is checked for collisions. The argument *i_squared*, is declared as REAL.

2.5.12.15.2 update_and_dead_reckon

This routine is used on "close" vehicles (those which are in the CIG buffer). It updates the copy of the vehicle's location in the CIG buffer and then dead reckons the vehicle.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh	register pointer to RVA_ENTRY	librva.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
veh_loc	register pointer to double	Standard
copy	register pointer to float	Standard
veh_copy	register pointer to MSG_OTHERVEH_STATE	/simnet/common/include/cig_if/sim_cig_if.h

Calls	
Function	**Where Described**
map_format_asid	Section 2.6.11.4.7

Table 2.5-282: update_and_dead_reckon Information.

2.5.12.15.3 delete_or_timeout

This is a small macro which deletes a vehicle from the hash table and the priority list if it is marked for deletion. Otherwise, it checks to see if the vehicle has timed out. If the vehicle has timed out, it is marked for deletion next tick.

Calls	
Function	**Where Described**
rva_delete_hash_entry	Section 2.5.12.11.12
rva_forget_about_vehicle	Section 2.5.12.3.1

Table 2.5-283: delete_or_timeout Information.

2.5.12.15.4 rotate_rwa_blades

This routine rotates the rotor and tail blades on a helicopter. Conceptually, the engine speed is used to determine how fast the blades should be turning (assume that engine rpm maps directly to blade speed with no step down).

Algorithmically, the turretAzimuth and gunElevation fields in the vehicleAppearancePacket are the most convenient places to keep rotation information. This is especially true since the CIG treats the rotor and tail blade rotations equivalently to turret azimuth and gun elevation. Since the turret and gun fields are 32-bit unsigned longs, the arithmetic is quite strightforward on machines that handle unsigned arithmetic correctly. On these machines, the change in rotor position is added or subtracted directly, and the machine underflows and overflows automatically, giving the correct result.

A bug in the compiler used by the Butterfly machine causes the machine to preserve the sign bit whenever possible. For these machines, right shift both the current and delta positions before doing the arithmetic, and then left shift back. While this results in a loss of precision, a few bits are not really important since the positions have 32 bits of precision when converting the unsigned values into signs. The only concern is how many bits to shift. It is best to shift the smallest number of bits that avoid interfering with the sign bit in order to avoid loss of precision. It is also desirable to set up the shifted numbers so addition can be done without overflowing into the sign bit, and that subtraction can be done without underflowing (in case underflow does not work correctly). To solve this, do the original shift right by 3 bits, and then to avoid arithmetic problems, set the number 3 bit in both the turretAzimuth and gunElevation. Setting this bit ensures no underflow on subtraction. If the bit is not used in a subtract (or if this is an addition), then it is shifted off when left shifting back by 3 bits.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
curr_veh	register pointer to RVA_ENTRY	librva.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
sine_ptr	register pointer to float	Standard
sg_otherveh	pointer to MSG_OTHERVEH_STATE	/simnet/common/include/cig_if/sim_cig_if.h
sin_cos_index	int	Standard
turn_amount	register double	Standard
rwa_type	ObjectType	/simnet/common/include/protocol/p_sim.h
circle_frac	unsigned long	Standard

Calls	
Function	**Where Described**
veh_get_force	libveh
SINES_SHIFT_INDEX	sines.h

Table 2.5-284: rotate_rwa_blades Information.

2.5.12.15.5 adjust_dynamic_vehicles

This routine does all the tick by tick processing of dynamic vehicles. It checks to see if vehicles need to be deleted or timed out, if new vehicles have arrived, if vehicles have gone out of visual range, etc. It does the remove vehicle approximation for each vehicle, and updates the location, orientation, and appearance of the vehicles in the CIG buffer. The last priority list whose vehicles get into the CIG buffer is called the cutoff list. The cutoff list is determined and communicated to libfilter.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
list_num	int	Standard
buf_veh_cnt	int	Standard
extra_veh_cnt	int	Standard
curr_list	register pointer to PRIORITY_LIST	Section 2.5.12.17, prior_loc.h
curr_veh	register pointer to RVA_ENTRY	librva.h
prev_veh	register pointer to RVA_ENTRY	librva.h
curr_minus_timeout	int	Standard
num_new_dynamics	int	Standard
i	register int	Standard

Calls	
Function	**Where Described**
timers_get_current_tick	Section 2.6.3.1.1
delete_or_timeout	Section 2.5.12.15.3
prior_debug	Macro defined in Section 2.5.12.17 prior_loc.h
is_air_vehicle	Section 2.6.10.1.1
is_rwa	Section 2.6.10.11.1
rotate_rwa_blades	Section 2.5.12.15.4
zero_dead_reckon	Section 2.5.12.29.5
update_and_dead_reckon	Section 2.5.12.15.2
check_very_close_veh	Section 2.5.12.15.1
filter_change_class_bound	Section 2.5.14.2.1
delete_veh_from_cig_msg	Section 2.1.2.2.2.27.1
rva_forget_about_vehicle	Section 2.5.12.3.1
add_veh_to_cig_msg	Section 2.1.2.2.2.1.1

Table 2.5-285: adjust_dynamic_vehicles Information.

2.5.12.15.6 adjust_static_vehicles

This routine does all of the tick by tick adjustments on static vehicles. For every vehicle on the static vehicles list, it checks to see if the vehicle needs to be added, deleted, changed, or removed from the list. It also checks for timeouts and resupply conditions.

Internal Variables		
Variable	Type	Where Typedef Declared
temp_veh	register pointer to RVA_ENTRY	librva.h
prev_veh	register pointer to RVA_ENTRY	librva.h
curr_minus_timeout	int	Standard

Calls	
Function	Where Described
timers_get_current_tick	Section 2.6.3.1.1.1
rva_delete_hash_entry	Section 2.5.12.11.12
rva_forget_about_vehicle	Section 2.5.12.3.1
prior_debug	Macro defined in Section 2.5.12.17 prior_loc.h
check_very_close_veh	Section 2.5.12.15.1

Table 2.5-286: adjust_static_vehicles Information.

2.5.12.16 prior_loc.c

(./simnet/release/src/libsrc/librva/prior_loc.c)

This file includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"prior_loc.h"

This file declares the following variables:

static_vehs
sorted_vehicles
num_priority_lists
static_list_num
smooth_cutoff
cutoff_list_for_rva
max_vehs_in_buf
curr_vehs_in_buf
extra_vehs_allowed
max_statics_allowed

2.5.12.17 prior_loc.h

(./simnet/release/src/libsrc/librva/prior_loc.h)

This file includes:

"librva.h"
"basic.h"
"libfilter.h"

This file defines the following macros and constants:

NUM_TYPES_PER_LIST
PRIOR_OPTIMIZE
MIN_VEHS_IN_BUF
prior_debug()

This file defines the following types and declares variables as those types:

veh_priority_list	PRIORITY_LIST

This file declares the following external variables:

static_vehs
sorted_vehicles
num_priority_lists
static_list_num
smooth_cutoff
cutoff_list_for_rva
max_vehs_in_buf
curr_vehs_in_buf
extra_vehs_allowed
max_statics_allowed

2.5.12.18 prior_rm.c

(./simnet/release/src/libsrc/librva/prior_rm.c)

This file includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"rva_loc.h"
"prior_loc.h"

2.5.12.18.1 try_to_remove_veh

This routine deletes the vehicle specified by *veh* from the priority list it is on.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh	pointer to RVA_ENTRY	librva.h
pri_list	pointer to PRIORITY_LIST	Section 2.5.12.27, prior_loc.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
tmp_veh	pointer to RVA_ENTRY	librva.h
Errors		
Error	**Reason for Error**	
PRINT_VID_ERROR	PANIC standard couldn't delete veh from priority lists	

Table 2.5-287: try_to_remove_veh Information.

2.5.12.19 prior_sort.c

(./simnet/release/src/libsrc/librva/prior_sort.c)

This file includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"simstdio.h"
"libhull.h"
"rva_loc.h"
"prior_loc.h"

This file defines the following macros:

get_range_to_veh
move_to_new_list

2.5.12.20 proc_update.c

(./simnet/release/src/libsrc/librva/proc_update.c)

This file includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"sines.h"
"mass_stdc.h"

"dgi_stg.h"
"sim_cig_if.h"
"pro_sim.h"
"assoc.h"
"timers.h"
"libveh.h"
"libmap.h"
"libhull.h"
"libkin.h"
"rva_loc.h"
"prior_loc.h"

2.5.12.20.1 process_known_static

This routine processes a vehicle appearance packet *(vap)* from a static vehicle which is currently in the RVA table. This routine cehcks to see if the vehicle's appearance has changed, requiring an update to libmsg.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
r	pointer to RVA_ENTRY	librva.h
vap	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h

Calls	
Function	**Where Described**
prior_debug	Macro defined in Section 2.5.12.17 prior_loc.h

Table 2.5-288: process_known_static Information.

2.5.12.20.2 process_unknown_static

This routine processes a vehicle appearance packet *(vap)* from a static vehicle which is not currently in the RVA table, places it into the RVA Table and prioritizes it.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
r	pointer to RVA_ENTRY	librva.h
vap	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
temp_veh	pointer to RVA_ENTRY	librva.h
hash_id	int	Standard

Calls	
Function	**Where Described**
rva_add_hash_entry	Section 2.5.12.11.13
VEHICLE_IDS_EQUAL	Macro defined in /simnet/common/include/global/sim_macros.h
timers_get_current_tick	Section 2.6.3.1.1.1
prior_debug	Macro defined in Section 2.5.12.17 prior_loc.h

Table 2.5-289: process_unknown_static Information.

2.5.12.20.3 process_known_dynamic

This routine processes a vehicle appearance packet *(vap)* from a dynamic vehicle which is currently in the RVA table. The vehicle is prioritized. This routine is not for a vehicle which is on priority list zero.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
r	pointer to RVA_ENTRY	librva.h
vap	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
sin_cos_index	register int	Standard
list	pointer to PRIORITY_LIST	Section 2.5.12.17, prior_loc.h
curr_list	int	Standard
new_list	int	Standard
r_squared	double	Standard
sine_ptr	register pointer to float	Standard

Errors	
Error	**Reason for Error**
PRINT_VID_ERROR	PANIC - invalid priority list for vehID

Calls	
Function	**Where Described**
kinematics_range_squared	Section 2.5.8.10.1
rva_get_priority_list	Section 2.5.12.6.1
prior_debug	Macro defined in Section 2.5.12.17 prior_loc.h
try_to_remove_veh	Section 2.5.12.18.1
is_rwa	Section 2.6.10.11.1
fvec_scale	Section 2.6.2.23
map_format_asid	Section 2.6.11.4.7
veh_get_force	Section 2.6.10.6.3
map_net_to_cig	Section 2.6.11.5.8
fmat_copy	Section 2.6.2.12.1
SINES_SHIFT_INDEX	sines.h
rva_forget_about_vehicle	Section 2.5.12.3.1

Table 2.5-290: process_known_dynamic Information.

2.5.12.20.4 process_unknown_dynamic

This routine processes a vehicle appearance packet *(vap)* from a dynamic vehicle which is not currently in the RVA table. The vehicle is placed in the RVA table and prioritized.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
r	pointer to RVA_ENTRY	librva.h
vap	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
list_num	int	Standard
list	pointer to PRIORITY_LIST	prior_loc.h
hash_id	int	Standard
r_squared	double	Standard

Calls	
Function	**Where Described**
kinematics_range_squared	Section 2.5.8.10.1
rva_get_priority_list	Section 2.5.12.6.1
prior_debug	Macro defined in Section 2.5.12.17 prior_loc.h
rva_add_hash_entry	Section 2.5.12.11.13
fvec_scale	Section 2.6.2.23
timers_get_current_tick	Section 2.6.3.1.1

Table 2.5-291: process_unknown_dynamic Information.

2.5.12.20.5 rva_process_update

This routine is called by **process_a_packet()** in libRcvNet to process a vehicle appearance packet. It determines if the vehicle is static or dynamic, known or unknown, or if it is on the special smoothing list (list zero).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vap	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
r	pointer to RVA_ENTRY	librva.h
hash_id	int	Standard

Calls	
Function	**Where Described**
rva_find_hash_entry	Section 2.5.12.11.11
process_unknown_static	Section 2.5.12.20.2
process_unknown_dynamic	Section 2.5.12.20.4
process_known_static	Section 2.5.12.20.1
fvec_scale	Section 2.6.2.23
zero_process_dynamic	Section 2.5.12.29.4
process_known_dynamic	Section 2.5.12.20.3
timers_get_current_tick	Section 2.6.3.1.1

Table 2.5-292: rva_process_update Information.

2.5.12.21 range_sqrd.c

(./simnet/release/src/libsrc/librva/range_sqrd.c)

This file includes:

"stdio.h"
"math.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"rva_loc.h"

2.5.12.21.1 cig_get_current_range_sqrd

This function gets the square of the current range.

Return Values		
Return Value	**Type**	**Meaning**
square(max_dgi_range)	REAL	The square of the current range.

Table 2.5-293: cig_get_current_range_sqrd Information.

2.5.12.22 rva_init.c
(./simnet/release/src/libsrc/librva/rva_init.c)

This file includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"pro_sim.h"
"rva_loc.h"
"prior_loc.h"

2.5.12.22.1 rva_init

This routine initializes the rva hash table, the marker hash table, the entries in the rva table, and the priority lists.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
this_veh	register pointer to HASH_ENTRY	Section 2.5.12.24, rva_loc.h
list	register pointer to PRIORITY_LIST	Section 2.5.12.17, prior_loc.h

Calls	
Function	**Where Described**
rva_init_rva_table	Section 2.5.12.11.10
rva_init_marker_table	Section 2.5.12.13 .2

Table 2.5-294: rva_init Information.

2.5.12.23 rva_loc.c
(./simnet/release/src/libsrc/librva/rva_loc.c)

This file includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"pro_sim.h"
"rva_loc.h"

This file defines:
RVA_DEBUG

This file declares:
rva_debug
air_veh_list[]
close_veh_list[]
static_veh_list[]
remove_veh_list[]
chg_veh_list[]
add_dynamic_veh_list[]
vehicleIDIrrelevant
num_air_vehs
num_close_vehs
close_ctr
num_static_vehs
num_to_remove
num_chg_vehs
max_dgi_range

2.5.12.24 rva_loc.h

(./simnet/release/src/libsrc/librva/rva_loc.h)

This file includes "librva.h".

This file defines the following constants and macros:
N_HASH_VALUES
N_HASH_ENTRIES
PRINT_VID_ERROR

This file defines the following types:
HASH_TABLE
HASH_ENTRY

This file declares the following external variables and procedures:
rva_debug
remote_vehicles
air_veh_list[]
close_veh_list[]
static_veh_list[]
remove_veh_list[]
chg_veh_list[]
num_air_vehs
num_close_vehs
num_static_vehs
num_to_remove
num_chg_vehs
max_dgi_range

deal_with_possible_collision()
try_to_remove_veh()
zero_process_dynamic()
zero_dead_reckon()

rva_init_hashing()
rva_alloc_marker_table()
rva_init_marker_table()
rva_alloc_rva_table()
rva_init_rva_table()
rva_init_hash_table()

2.5.12.25 rva_setup.c
(./simnet/release/src/libsrc/librva/rva_setup.c)

This file includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"pro_sim.h"
"rva_loc.h"

2.5.12.25.1 rva_setup

This routine allocates the RVA table and Markers table, and calls **rva_priority_setup()**. Note that the setup only happens once, while initializations may happen any number of times.

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

Calls	
Function	**Where Described**
rva_alloc_rva_table	Section 2.5.12.11.9
rva_alloc_marker_table	Section 2.5.12.13.1
rva_priority_setup	Section 2.5.12.14.1

Table 2.5-295: rva_setup Information.

2.5.12.26 show_vehs.c
(./simnet/release/src/libsrc/librva/show_vehs.c)

This file is used for the stealth vehicle in order to keep the vehicle it is attached to invisible to the stealth. Information about the attached vehicle is not put into the CIG buffer.

This file includes:
"stdio.h"
"sim_dfns.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"rva_loc.h"

"libhull.h"
"libkin.h"
"prior_loc.h"

2.5.12.26.1 rva_vehicle_is_visible

This routine is called to turn a vehicle visible.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
r	pointer to RVA_ENTRY	librva.h
list_num	int	Standard
hash_id	int	Standard
Errors		
Error	**Reason for Error**	
PRINT_VID_ERROR	vehicle is unkown	
Calls		
Function	**Where Described**	
rva_find_hash_entry	Section 2.5.12.11.11	

Table 2.5-296: rva_vehicle_is_visible Information.

2.5.12.26.2 rva_vehicle_is_invisible

This routine is called to turn a vehicle invisible. Note that this vehicle will be maintained in the RVA table forever (no matter what the range) until it either becomes visible or is removed by timeout. If a static vehicle is made invisible, then the routine in "tell_cig.c" will add the vehicle to the remove list.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
r	pointer to RVA_ENTRY	librva.h
hash_id	int	Standard
Errors		
Error	**Reason for Error**	
PRINT_VID_ERROR	vehicle is unkown	
Calls		
Function	**Where Described**	
rva_find_hash_entry	Section 2.5.12.11.11	
delete_veh_from_cig_msg	Section 2.1.2.2.2.27.1	

Table 2.5-297: rva_vehicle_is_invisible Information.

2.5.12.26.3 vehicle_is_visible

This routine allows either **rva_vehicle_is_visible**() or **vehicle_is_visible**() to be used interchangeably (for backwards compatibility with existing code).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh	pointer to VehicleID	/simnet/common/include/protocol/basic.h
Calls		
Function	**Where Described**	
rva_vehicle_is_visible	Section 2.5.12.26.1	

Table 2.5-298: vehicle_is_visible Information.

2.5.12.26.4 vehicle_is_invisible

This routine allows either **rva_vehicle_is_invisible()** or **vehicle_is_invisible()** to be used interchangeably (for backwards compatibility with existing code).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh	pointer to VehicleID	/simnet/common/include/protocol/basic.h

Calls	
Function	**Where Described**
rva_vehicle_is_invisible	Section 2.5.12.26.2

Table 2.5-299: vehicle_is_invisible Information.

2.5.12.27 tell_cig.c
(./simnet/release/src/libsrc/librva/tell_cig.c)

This file includes:
"stdio.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"rtc.h"
"libcig.h"
"rva_loc.h"
"prior_loc.h"

This file declares the following external procedures:
adjust_dynamic_vehicles()
adjust_static_vehicles()
adjust_markers()

2.5.12.27.1 rva_tell_cig_about_other_vehicles

This routine initiates the tick by tick processing by calling the adjust vehicles and markers routines. This routine communicates with the CIG buffer and reduces or increments the number of vehicles in the CIG buffer if the CIG reports that it cannot keep up.

Calls	
Function	**Where Described**
prior_debug	Macro defined in Section 2.5.12.17 prior_loc.h
rtc_start_time	Section 2.6.16.1.2
adjust_dynamic_vehicles	Section 2.5.12.15.5
adjust_static_vehicles	Section 2.5.12.15.6
adjust_markers	Section 2.5.12.13.5
rtc_stop_time	Section 2.6.16.1.3

Table 2.5-300: rva_tell_cig_about_other_vehicles Information.

2.5.12.28 too_many_vehs.c
(./simnet/release/src/libsrc/librva/too_many_vehs.c)

This file includes:
"stdio.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"rva_loc.h"
"prior_loc.h"

2.5.12.28.1 cig_too_many_vehicles

The CIG interface code calls this routine when the CIG returns an overload message. The number of vehicles that are allowed into the buffer is decremented by 2. This routine applies only to dynamic vehicles.

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

Table 2.5-301: cig_too_many_vehicles Information.

2.5.12.29 zero_veh.c
(./simnet/release/src/libsrc/librva/zero_veh.c)

These routines are applicable to the Stealth vehicle for smoothing the movement of the attached vehicle. These routines implement a smoothing algorithm for vehicles which need to be dead reckoned with a very high level of fidelity. This algorithm is used on vehicles which have been locked into the priority lists by having been placed on priority list zero.

When a new vehicle appearance packet is received for the vehicle, the vehicle is dead reckoned forward by the number of ticks specified in *smooth_ticks*. Then, an alternate velocity vector for the vehicle is calculated which takes it from its present (dead reckoned) position to the extrapolated position. Similarly, a rotation matrix is created which smoothly reorients the vehicle to its correct (as specified in the new vehicle appearance packet) orientation over time. Instead of dead reckoning by adding the velocity vector, the alternate velocity is used with a rotation update until the number of ticks speicfied in *smooth_ticks* have passed. Then, the velocity specified in the packet used with no rotation updates.

Note that this is still a rather expensive algorithm, and should not be run for more than a very few vehicles at any one time. Currently, only one vehicle can be put into list zero at a time.

This file includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stg.h"
"sim_cig_if.h"
"sines.h"
"simstdio.h"
"timers_dfn.h"
"timers.h"
"libkin.h"
"libveh.h"
"rva_loc.h"
"prior_loc.h"

The following declarations are made:

spec_dead_reckon
smooth_ticks
pos_veloc[3]
orient_veloc[3] [3]
turret_az_vel
turret_el_vel
true_turret_az
true_turret_el

The following constants are defined:

MIN_SMOOTH_TICKS
DIST_THRESH

2.5.12.29.1 zero_init_veh

This routine is called to set the hash ID on the zero priority list. Currently, only one vehicle is allowed on this list.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
hash_id	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOL	only one vehicle can be zeroed at a time.
FALSE	BOOL	hash ID is initiated on the zero priority list

Table 2.5-302: zero_init_veh Information.

2.5.12.29.2 zero_uninit_veh

This routine is called to indicate that the specified vehicle has been removed from the zero priority list.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
hash_id	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOL	invalid hash ID
FALSE	BOOL	hash ID is removed from the zero priority list

Table 2.5-303: zero_uninit_veh Information.

2.5.12.29.3 zero_get_new_velocities

This routine calculates what the temporary position and orientation velocities should be. It writes new values into all fields of the smooth vehicle structure.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
r	pointer to RVA_ENTRY	librva.h
vap	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
curr_veloc	array 3 of float	Standard
reckd_pos	array 3 of double	Standard
tmp_mat	3 by 3 matrix of float	Standard
delta_orient	3 by 3 matrix of float	Standard
mat_ptr	array 3 of float	Standard
heading	float	Standard
pitch	float	Standard
roll	float	Standard
turr_az	float	Standard
turr_el	int	Standard
inv_t_sqrd	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
timers_get_current_tick	Section 2.6.3.1.1
vec_copy	Section 2.6.2.59
fvec_copy	Section 2.6.2.26 .1
fmat_copy	Section 2.6.2.12.1
fmat_transpose	Section 2.6.2.19.1
fmat_mat_mul	Section 2.6.2.14
fmat_rot_init2	Section 2.6.2.17.1

Table 2.5-304: zero_get_new_velocities Information.

2.5.12.29.4 zero_process_dynamic

This routine is called when a vehicle appearance packet is received from a vehicle on priority list zero. After making a call to get new position and orientation updates, it copies all of the other information into the CIG buffer copy of the vehicle.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
r	pointer to RVA_ENTRY	librva.h
vap	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
reckoned_orient	3 by 3 matrix of float	Standard
reckoned_pos	arrary 3 of REAL	/simnet/common/include/global/sim_types.h
diff	array 3 of REAL	/simnet/common/include/global/sim_types.h
thresh	REAL	/simnet/common/include/global/sim_types.h
reckoned_azimuth	Angle	/simnet/common/include/protocol/basic.h
reckoned_elevation	Angle	/simnet/common/include/protocol/basic.h
msg_otherveh	pointer to MSG_OTHERVEH_STATE	/simnet/common/include/cig_if/sim_cig_if.h
sine_ptr	register pointer to float	Standard
sin_cos_index	int	Standard

Calls	
Function	**Where Described**
vec_sub	Section 2.6.2.65
vec_mag3	/simnet/common/include/global/sim_macros.h
zero_get_new_velocities	Section 2.5.12.29.3
fmat_copy	Section 2.6.2.12.1
vec_copy	Section 2.6.2.59
map_format_asid	Section 2.6.11.4.7
veh_get_force	Section 2.6.10.6.3
map_net_to_cig	Section 2.6.11.5.8
SINES_SHIFT_INDEX	sines.h

Table 2.5-305: zero_process_dynamics Information.

2.5.12.29.5 zero_dead_reckon

This routine dead reckons a vehicle on the zero priority list.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
r	pointer to RVA_ENTRY	librva.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
veh_loc	pointer to float	Standard
copy_loc	pointer to float	Standard
tmp	pointer to MSG_OTHERVEH_STATE	/simnet/common/include/cig_if/sim_cig_if.h
saved_align	unsigned char	Standard

Errors	
Error	**Reason for Error**
PRINT_VID_ERROR	Can't adjust veh app of invalid veh_id
Calls	
Function	**Where Described**
d2f_vec_copy	Section 2.6.2.2.1
fmat_copy	Section 2.6.2.12.1
fmat_mat_mul	Section 2.6.2.14
fvec_copy	Section 2.6.2.26.1
fvec_add	Section 2.6.2.25.1

Table 2.5-306: zero_dead_reckon Information.

2.5.12.29.6 zero_set_extrapolation_period

This routine is used to set the number of ticks over which the vehicle is restored from a known incorrect position and orientation to a predicted correct position and orientation. This void procedure takes a single argument, *num_ticks,* which is declared as an int.

2.5.13 librva_util
(./simnet/release/src/libsrc/librva_util [librva_util])

2.5.13.1 get_list.c
(./simnet/release/src/libsrc/librva_util/get_list.c)

Includes:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"p_sim.h"
"libmatrix.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"librva.h"
"librva_util.h"

2.5.13.1.1 rva_create_output_list

This routine is a stub in anticipation of release 6.6.1 routines.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
inclusion_fn()	pointer to function that returns int	Standard

Table 2.5-307: rva_create_output_list Information.

2.5.13.1.2 rva_get_output_list

This routine provides an interface to the librva routines by getting either a close vehicles list or an air vehicles list from the librva hash table, given the #define input parameter, *list_id*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
list_id	int	Standard
list	pointer to pointer to pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
num_vehicles	pointer to int	Standard
Return Values		
Return Value	**Type**	**Meaning**
1	int	the procedure was successful
Calls		
Function	**Where Described**	
rva_get_close_list	Section 2.5.12.9.1	
rva_get_air_veh_list	Section 2.5.12.4.1	

Table 2.5-308: rva_get_output_list Information.

2.5.13.1.3 rva_util_get_veh_app_pkt

This routine calls the librva routine, rva_get_veh_app_pkt().

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vehicle	pointer to VehicleID	/simnet/common/include/protocol/basic.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
hash_id	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
rva_get_veh_app_pkt	pointer to VehicleAppearanceVariant	the vehicle appearance variant
0	int	Vehicle is unknown
Calls		
Function	**Where Described**	
rva_get_veh_app_pkt	Section 2.5.12.7.1	

Table 2.5-309: rva_util_get_app_pkt Information.

2.5.13.2 librva_util.h
(./simnet/release/src/libsrc/librva_util/librva_util.h)

Includes:

"sim_types.h"
"p_sim.h"

External Procedure Declarations:

rva_create_output_list()
rva_get_output_list()
rva_smooth_vehicle()
rva_dont_smooth_vehicle()
rva_util_get_veh_app_pkt()

This file contains function declarations and definitions used in librva_util.

2.5.14 libfilter

(./simnet/release/src/libsrc/libfilter [libfilter])

In an exercise with a large number of simulated vehicles, most VAPs are of no interest to a given simulator, because they are not one of the N highest priority vehicles. Libfilter provides librva with an interface to download the necessary data to the network device to discard unwanted VAPs at the network device level. This process is called filtering. Network filtering results in the elimination of unnecessary data transfers between the network device and host memory, which can be prohibitively expensive. The M1 and M2 Masscomp and Simnet Butterfly machines all have CMC cards which connect to the network. Each CMC card has its own processor, and in an effort to reduce the number of vehicles, some processing is offloaded to the CMC processor where out of range vehicles will be filtered out. Two shared memory degments allow data to go back and forth between applications and host using pointers. The files in this code contain routines for communicating with the program running on the CMC card (except for filter.c)

2.5.14.1 add.c

(./simnet/release/src/libsrc/libfilter/add.c)

This file contains the routines to specify the filter class. The file includes "filter_loc.h".

2.5.15.1.1 filter_add_class

This routine adds a class of vehicles to the filter list, for example, air vehicles beyond a certain range will be filtered. The routine takes as parameters: *class_num* -- the class number, *obj_class* -- a pointer to the object class being added, *alignment* (or alignments) to be included, and the *range* of vehicles in this class. This routine returns -1 if *class_num* is out of range, and 0 otherwise.

Parameters

Parameter	Type	Where Typedef Declared
class_num	int	Standard
obj_class	pointer to REMOTE_OBJ_CLASS	libfilter.h
alignment	int	Standard
range	int	Standard

Internal Variables

Variable	Type	Where Typedef Declared
numclasses	register int	Standard

Return Values

Return Value	Type	Meaning
0	int	procedure was successful
-1	int	class number is out of range; procedure failed

Calls

Function	Where Described
DATACOPY	Macro defined in filter_loc.h

Table 2.5-310: filter_add_class Information.

2.5.14.2 bounds.c

(./simnet/release/src/libsrc/libfilter/bounds.c)

Includes:
"filter_loc.h"

2.5.14.2.1 filter_change_class_bound

This routine is used to change the boundary on a filter class. It takes as parameters: *class_num* -- the class number, the *alignment*, and a *range*. If the alignment is specified as ALIGHNED_BOTH, then the new range applies to both friend and foe boundaries, otherwise, it applies to the specified boundary only. The new boundaries are downloaded to the CMC card. As with **filter_add_class**(), this routine saves the new range for computing boundaries when the filter location changes. The routine returns -1 if *class_num* is out of range, and 0 otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
class_num	int	Standard
alignment	int	Standard
range	int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
box	register pointer to BOUND_BOX	filter_loc.h

Return Values		
Return Value	**Type**	**Meaning**
0	int	procedure was successful
-1	int	class number is out of range; procedure failed

Calls	
Function	**Where Described**
DATACOPY	Macro defined in filter_loc.h

Table 2.5-311: filter_change_class_bound Information.

2.5.14.3 data.c

(./simnet/release/src/libsrc/libfilter/data.c)

Includes:
"filter_loc.h"

This file defines all data shared between the various libfilter functions.

Variable Declarations:

bound_info -- a local image of what will be downloaded to the CMC card.
card_bound_info -- a pointer to the *bound_info* portion of the CMC card memory.
class_ranges[] -- contains the current range for each class; since this changes infrequently, it does not need to be passed around

host_info -- a pointer to the shared memory on the host containing information about the different classes being filtered on.

filter_location -- the center of all the bound boxes, twice the range on each side; currently the z coordinate is ignored.

2.5.14.4 dump.c

(./simnet/release/src/libsrc/libfilter/dump.c)

Includes:

"network.h"
"filter_loc.h"

This file contains all definitions needed to display the current filter information on the CMC card.

2.5.14.4.1 filter_dump_filter_info

This routine puts information about the filter currently on the CMC card onto stdout. The [NUMBER_OF_REGISTERS - 1] register (the last register) is used for the maximum CIG range box.

Internal Variables		
Variable	Type	Where Typedef Declared
i	register int	Standard
limits	register pointer to VEH_LIMITS	libfilter.h
class	register pointer to REMOTE_OB_CLASS	libfilter.h
rejects	int	Standard

Calls	
Function	Where Described
net_reg_read	Section 2.20.2.16.1 See MCC CSCI document
network_get_net_handle	Section 2.1.1.3.2.12.1

Table 2.5-312: filter_dump_filter_info Information.

2.5.14.5 filter.c

(./simnet/release/src/libsrc/libfilter/filter.c)

Includes:

"enpif.h"
"network.h"
"p_assoc.h"
"p_sim.h"
"p_num.h"
"veh_type.h"
"libfilter.h"
"hod_to_i.h"

This file contains application specific routines which will be linked into the CMC ethernet driver to filter packets on the network.

The macro CHECK_RANGE() is defined. Note that this causes a return of FREE_PACKET if the *x* and *y* given are outside the box.

2.5.14.5.1 do_packet_from_network

This routine gets called by the CMC driver when a packet arrives. The routine decides if the CMC card should put the packet into the host's ring buffer by returning SEND_PACKET or discard the packet by returning FREE_PACKET. A set of vehicle classes are downloaded to the card when it is initialized, and boxes which describe the acceptable range for vehicles in the classes are downloaded from the host as necessary. Note that non-vehicle appearance packets are accepted immediately. Appearance packets are rejected if they fall outside the maximum CIG range. Vehicle appearance packets are only accepted if the vehicle is within the box specified for its class (and alignment) or if it does not fall into any specified class. Note that a vehicle may fall into a class but not have a box set for its alignment. In this case, the routine continues to look for a box that the vehicle may fall into. *class* is the pointer to the current vehicle class. *box* is the pointer to the current test area. *i* is the loop counter for classes. *pdu_x* and *pdu_y* are the location of the vehicle as integers. The routine declares pointers into the various parts of the packet. *host_info* is the number of classes and list of classes. *bound_info* contains the force id and bounding boxes.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to short	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
class	register pointer to REMOTE_OBJ_CLASS	*libfilter.h*
box	register pointer to BOUND_BOX	filter_loc.h
high_order_double	register long	Standard
i	register int	Standard
vap	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
mp	pointer to MarkerVariant	/simnet/common/include/protocol/p_sim.h
pdu_x	long	Standard
pdu_y	long	Standard
buf	pointer to NetworkPacket	libnetif
apdu	pointer to AssociationPDU	/simnet/common/include/protocol/p_assoc.h
spdu	pointer to SimulationPDU	/simnet/common/include/protocol/p_sim.h
host_info	pointer to HOST_INFO	filter_loc.h
bound_info	pointer to BOUND_INFO	filter_loc.h
veh_limits	pointer to VEH_LIMITS	filter_loc.h
Return Values		
Return Value	**Type**	**Meaning**
SEND_PACKET	int	packet is sent
FREE_PACKET	int	packet is freed; coordinates are outside the range

Calls	
Function	**Where Described**
GET_DATA_PTR	Macro defined in network.h
HIGH_ORDER_DOUBLE_TO_INT	Macro defined in hod_to_i.h
CHECK_RANGE	Macro defined in filter.c

Table 2.5-313: do_packet_from_network Information.

2.5.14.5.2 do_packet_from_host

This routine is called everytime the CMC driver receives a packet from the host. The routine determines if the packet should be sent. SEND_PACKET is returned if the packet should be sent and FREE_PACKET is returned if the packet should be ignored. Note that currently all packets from the host are sent.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to short	Standard
Return Values		
Return Value	**Type**	**Meaning**
SEND_PACKET	int	The packet is placed into the ring buffers

Table 2.5-314: do_packet_from_host Information.

2.5.14.5.3 do_init

This routine is called by the CMC driver when the card is initialized (either on a reboot or when **net_stop**() is called followed by **net_norm**()). *host_info* is the number and list of classes. *bound_info* is the force id and bounding boxes.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
host_info	pointer to HOST_INFO	filter_loc.h
bound_info	pointer to BOUND_INFO	filter_loc.h

Table 2.5-315: do_init Information.

2.5.14.6 force.c
(./simnet/release/src/libsrc/libfilter/force.c)

Includes:
"filter_loc.h"

This file contains all definitions needed to inform the CMC card of a change in the host's force id.

2.5.14.6.1 filter_set_force

This routine takes a force id, *force*, and writes its value to the CMC card via the *host_info* shared memory pointer.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
force	ForceID	/simnet/common/include/protocol/basic.h

Table 2.5-316: filter_set_force Information.

2.5.14.7 init.c
(./simnet/release/src/libsrc/libfilter/init.c)

Includes:
"network.h"
"filter_loc.h"

This file contains all definitions needed to initialize filtering packets on the net.

2.5.14.7.1 filter_init

This routine initializes the filter. The *min_x* for undefined bound boxes and *num_classes* start at zero, and *card_bound_info* and *host_info* get initialized. This routine must be called AFTER **network_init**(). This routine returns -1 on failure, and 0 otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
handle	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
class_num	register int	Standard
range	register pointer to CLASS_RANGES	filter_loc.h
class	REMOTE_OBJ_CLASS	libfilter.h
limit	register pointer to VEH_LIMITS	filter_loc.h
hb_size	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
0	int	procedure was successful
-1	int	procedure failed

Errors	
Error	**Reason for Error**
perror	filter_hostbuf failure
Calls	
Function	**Where Described**
net_hostbuf_info	Section 2.20.2.6.1 MCC CSCI document
DATA_COPY	Macro defined in filter_loc.h
net_reg_write	Section 2.20.2.16.2

Table 2.5-317: filter_init Information.

2.5.14.8 location.c

(./simnet/release/src/libsrc/libfilter/location.c)

Includes:
"filter_loc.h"

This file contains all definitions needed change the location around which classes are filtered.

Variable Declarations:

filter_threshold -- rather than downloading new boxes to the card every time the location changes slightly, a threshold is defined. New bound boxes are downloaded to the CMC card only when the location has changed in either the x or y direction by the threshold amount. The threshold is intially set for 100 meters, but can be changed dynamically using **filter_set_filter_threshold()**.

max_cig_range -- Before checking individual classes for inclusion in a bounding box, every vehicle appearance packet is tested against a single maximum, typically the maximum range at which the CIG can handle vehicles. This is initially set at 3500 meters, but can be changed dynamically.

2.5.14.8.1 filter_set_filter_threshold

This routine sets the filter threshold to the value of the input parameter *threshold.*

2.5.14.8.2 filter_set_max_cig_range

This routine sets the maximum CIG range.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
range	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
box	*register pointer to* BOUND_BOX	*filter_loc.h*
Calls		
Function	**Where Described**	
DATACOPY	Macro defined in filter_loc.h	

Table 2.5-318: filter_set_max_cig_range Information.

2.5.15 libimpacts

(./simnet/release/src/libsrc/libimpacts [libimpacts])

One CSU, libimpacts provides routines for managing impact information received from the network. Time-delayed impacts are scheduled and executed from this module. Libimpacts also contains functionality to inform libmsg of impacts to be displayed by the CIG.

2.5.15.1 impacts.c

(./simnet/release/src/libsrc/libimpacts/impacts.c)

Includes:
"stdio.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"libsound.h"
"libsound_dfn.h"
"libmap.h"
"libimps.h"

This file defines the structure type IMPACT as follows:

ammo_type	-- the type of ammo round
imp_type	-- the impact type (as defined in 2.5.15.2 libimps.h)
loc	-- where the impact occurred
delay	-- the delay before impacting the ground (for indirect fire)
r_2	-- the range squared
last	-- the previous element in the linked impact list
next	-- the next element in the linked impact list

Variable and Procedure Declarations:
impact_array[MAX_EFF]
impact_free[MAX_IFF]
impact_list_start
impact_free_index
impacts_debug
impact_get_element()
impacts_free_element()

2.5.15.1.1 impacts_init

This routine initializes the impacts lists.

Internal Variables		
Variable	Type	Where Typedef Declared
i	int	Standard

Table 2.5-319: impacts_init Information.

2.5.15.1.2 impacts_tell_cig_about_impacts

This routine contains the linked impact lists with the associated delays. This routine handles the impact delay timing. The CIG and sound systems are notified of the imp_type, ammo_type, and r_2 elements of the impact structure parameter, *qe*, in order to produce the proper effects and sounds. Once the CIG and sound systems have been notified, the element is freed (note that the information stored in *qe* is still valid, it has just been taken out of the queued effects list and put on the free list).

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
qe	pointer to IMPACT	impacts.c
unique_identifier	int	Standard
i	int	Standard
cig_loc	R4P3D	/simnet/common/include/global/dgi_stdg.h

Calls	
Function	**Where Described**
map_get_burst_ground_from_ammo_entry	Section 2.6.11.2.4
cig_msg_prepend_show_effect	Section 2.1.2.2.2.96.1
sound_of_weapons_impact	Section 2.1.3.1.1
map_get_burst_air_from_ammo_entry	Section 2.6.11.2.5
map_get_burst_armor_from_ammo_entry	Section 2.6.11.2.6
map_get_burst_wood_from_ammo_entry	Section 2.6.11.2.7
map_get_burst_other_from_ammo_entry	Section 2.6.11.2.8
map_get_muzzle_flash_me_from_ammo_entry	Section 2.6.11.2.10
map_get_muzzle_flash_other_from_ammo_entry	Section 2.6.11.2.11
impacts_free_element	Section 2.5.15.1.5

Table 2.5-320: impacts_tell_cig_about_impacts Information.

2.5.15.1.3 impacts_queue_effect

This routine builds the impact structure from the input parameters, *ammo_type*, *imp_type*, *loc*, *delay*, and *r_2*, and then places the structure into the effects queue.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_type	int	Standard
imp_type	int	Standard
loc	VECTOR	/simnet/common/include/global/sim_types.h
delay	int	Standard
r_2	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
cig_get_current_range_sqrd	Section 2.5.12.21.1
impacts_get_element	Section 2.5.14.1.4

Table 2.5-321: impacts_queue_effect Information.

2.5.15.1.4 impacts_get_element

This routine tries to get a free pointer to an element in the impact list. If the pointer is unavailable, the routine returns FALSE. If the pointer is available, the routine returns TRUE and puts an empty element at the beginning of the impact list.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	pointer to element is available
FALSE	int	pointer is unavailable

Table 2.5-322: impacts_get_element Information.

2.5.15.1.5 impacts_free_element

This routine puts the element, *element*, back on the free list when complete.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to IMPACT	impact.c

Table 2.5-323: impacts_free_element Information.

2.5.15.2 libimps.h

Includes:

Defines the impact types as follows:

IMPACT TYPE	REFERENCE
IMPACT_NO_IMPACT	0
IMPACT_GROUND	1
IMPACT_AIR	2
IMPACT_ARMOR	3
IMPACT_WOOD	4
IMPACT_OTHER	5
IMPACT_US	6
IMPACT_MUZZLE_FLASH_ME	7
IMPACT_MUZZLE_FLASH_OTHER	8
IMPACT_NO_SOUND_RANGE	-1.0

Procedure Declarations:

impacts_init()
impacts_tell_cig_about_impacts()
impacts_queue_effect()

2.5.16 libapp

(./simnet/release/src/libsrc/libapp [libapp])

The simulation host periodically informs other entities on the network of its current location, orientation, and appearance. It uses routines in libapp to determine, each frame, if it necessary to place an update onto the network. Libapp checks for changes in appearance, changes in position or orientation which deviate from the dead reckoned model of the vehicle, or time-out. If the vehicle's appearance has changed, or an update has not been sent in the last 5 seconds, one will be generated. If an update is to be sent, libapp calls routines in "m1_network.c", "m2_network.c", or "m2_tracks.c" to fill in vehicle specific fields. Routines are called in m1_tracks.c to determine what size dust cloud, if any, should be reported.

2.5.16.1 libapp.h

This file defines the interface to the libapp library of vehicle appearance related software.

VehicleAppearance typedef consists of the vehicle's appearance at a moment in time:

time	-- the particular moment, in seconds, of the appearance
vehicleClass	-- class of vehicle
appearance	-- type of vehicle and appearance
rotation	-- pointer to rotation matrix
location	-- pointer to location
velocity	-- pointer to velocity vector
turretAzimuth	-- turret/hull orientation
gunElevation	-- gun/turret elevation

DiscrepanceThresholds typedef applied to a class of vehicles consists of:

locationThresh[3]	-- location threshold
rotationThresh	-- rotation angle about any axis
turretAzimuthThresh	-- turret rotation angle
gunElevationThresh	-- gun elevation angle

Macro and constant defines:

SIMNET_TO_RADIANS_FACTOR
simnet_angle_to_radians()
radians_to_simnet_angle()

Declarations of library routines:

ReadDiscrepancyThresholds()
PrepareDiscrepancyThresholds()
AppearanceDiscrepancyExceedsThresholds()
clear_monitor_variables()
get_reason_time()
get_reason_app()
get_reason_tur_azi()
get_reason_gun_elev()
get_reason_loc()
get_reason_rot()

2.5.16.2 read.c

This file contains code for reading a file of discrepancy thresholds. The file of threshold values contains the threshold parameters which, if exceeded, determine whether a VehicleAppearance Packet is sent to the network.

The format of a file of threshold values is:

dimensions	<width> <length> <height>
location	<location threshold in percent of vehicle dimension>
rotation	<rotation threshold in degrees>
turret-azimuth	<turret azimuth threshold in degrees>
gun-elevation	<gun elevation threshold in degrees>

Includes:
"stdio.h"
"simstdio.h"
"libapp.h"

Static variable declarations:
filename
file

External procedure declarations:
ReadThreshold()

2.5.16.2.1 ReadDiscrepancyThresholds

This routine reads a set of thresholds from an ASCII file. The caller provides the name of the file, *fname*, and storage for the thresholds, *thresholds*. The routine returns 1 if successful, and 0 otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fname	pointer to char	Standard
thresholds	pointer to DiscrepancyThresholds	libapp.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
dimensions	array 3 of double	Standard
location	array 3 of double	Standard
rotation	double	Standard
turretAzimuth	double	Standard
gunElevation	double	Standard
Return Values		
Return Value	**Type**	**Meaning**
0	int	procedure failed
1	int	procedure was successful

Calls	
Function	**Where Described**
ReadThreshold	Section 2.5.16.2.2
PrepareDiscrepancyThreshold	Section 2.5.16.3.1

Table 2.5-324: ReadDiscrepancyThresholds Information.

2.5.16.2.2 ReadThreshold

This routine is used to parse a single threshold parameter.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
format	pointer to char	Standard
number	int	Standard
value1	pointer to double	Standard
value2	pointer to double	Standard
value3	pointer to double	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
buffer	array 100 of char	Standard
Return Values		
Return Value	**Type**	**Meaning**
0	int	procedure failed
1	int	procedure was successful
Errors		
Error	**Reason for Error**	
stderr	Missing parameter in file	

Table 2.5-325: ReadThreshold Information.

2.5.16.3 thresh.c

This file contains code for measuring the discrepancy between two versions of a vehicle's appearance, and testing that discrepancy against a set of thresholds. The librva location and orientation approximations are compared to the vehicle's actual location and orientation. If the variance between the two exceeds the discrepency threshold values, a VehicleAppearance Packet is sent.

Includes:
    "math.h"
    "sim_dfns.h"
    "pro_sim.h"
    "pro_timers.h"
    "libapp.h"

Defines:
    register

Declarations for monitoring threshold discrepancies:
    reson_time
    reson_app
    reason_tur_azi
    reason_gun_elev
    reason_loc[3]
    reason_rot

2.5.16.3.1 PrepareDiscrepancyThreshold

This routine precomputes certain values associated with a set of thresholds.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
thresholds	register pointer DiscrepancyThresholds	libapp.h
dimensions	array 3 of double	Standard
location	array 3 of double	Standard
rotation	double	Standard
turretAzimuth	double	Standard
gunElevation	double	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
Calls		
Function	**Where Described**	
radians_to_simnet_angle	libapp.h	

Table 2.5-326: PrepareDiscrepancyThreshold Information.

2.5.16.3.2 AppearanceDiscrepancyExceedsThresholds

This routine compares two descriptions of a vehicle's appearance, a dead reckoned location and an exact location, to determine whether they differ by more than threshold amounts. The routine returns 1 if an update must be sent, and 0 otherwise. An update is sent because either:

1. Five seconds have elapsed since the last update.
2. Any appearance modifier bits have changed.
3. The turret azimuth discrepancy exceeds the threshold.
4. The gun elevation discrepancy exceeds the threshold.
5. The difference between the current location and the dead reckoned location exceeds the threshold.
6. The vehicle rotation discrepancy exceeds the threshold.

Parameters are represented as follows:

thresholds -- A pointer to the thresholds for the vehicle class.
lastUpdate -- A pointer to the last appearance broadcast.
currentApp -- A pointer to the current VAP.
dT -- The number of seconds since the last PDU was sent.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
thresholds	pointer to DiscrepancyThresholds	libapp.h
lastUpdate	register pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
currentApp	register pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
dT	double	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
row	register pointer to float	Standard
disc	register double	Standard
i	register long	Standard
thresh	array of double	Standard
position	array of 3 doubles	Standard
old_rot	pointer to float	Standard
new_rot	pointer to float	Standard
Return Values		
Return Value	**Type**	**Meaning**
0	int	the dead reckoned location may be used
1	int	an update must be send with the exact location

Table 2.5-327: AppearanceDisrepancyExceedsThresholds Information.

2.5.16.3.3 clear_monitor_variables

This routine sets the monitor variables to 0.

2.5.16.3.4 get_reason_time

This routine returns the number of times an update was sent due a time discrepancy.

Return Values		
Return Value	**Type**	**Meaning**
reason_time	unsigned long	the number of times an update was sent due to a time discrepancy

Table 2.5-328: get_reason_time Information.

2.5.16.3.5 get_reason_app

This routine returns the number of times an update was sent due to any appearance modifier bits discrepancy.

Return Values		
Return Value	**Type**	**Meaning**
reason_app	unsigned long	the number of times an update was sent to to an appearance discrepancy

Table 2.5-329: get_reason_app Information.

2.5.16.3.6 get_reason_tur_azi

This routine returns the number of times an update was sent due to a turret azimuth discrepancy.

Return Values		
Return Value	**Type**	**Meaning**
reason_tur_azi	unsigned long	the number of times an update was sent due to a turret azimuth discrepancy

Table 2.5-330: get_reason_tur_azi Information.

2.5.16.3.7 get_reason_gun_elev

This routine returns the number of times an update was sent due to a gun elevation discrepancy.

Return Values		
Return Value	**Type**	**Meaning**
reason_gun_elev	unsigned long	the number of times an update was sent due to a gun elevation discrepancy

Table 2.5-331: get_reason_gun_elev Information.

2.5.16.3.8 get_reason_loc

This routine returns the number of times an update was sent due to a location discrepancy.

Return Values		
Return Value	**Type**	**Meaning**
reason_loc	pointer to unsigned long	the number of times an update was sent due to a location discrepancy

Table 2.5-332: get_reason_loc Information.

2.5.16.3.5 get_reason_rot

This routine returns the number of times an update was sent due to a rotation discrepancy.

Return Values		
Return Value	**Type**	**Meaning**
reason_rot	unsigned long	the number of times an update was sent due to a rotation discrepancy

Table 2.5-333: get_reason_rot Information.

2.5.16.3.6 print_reasons

This routine prints the reasons for trasmitting packets. This routine is used for debugging purposes.

2.5.17 libnear
(./simnet/release/src/vehicle/libsrc/libnear [libnear])

Libnear is a utility library used by libmissile which contains routines which find the closest vehicle to a given point or vector. The library keeps track of the closest vehicle from tick to tick and attempts to stay locked on to that vehicle if possible.

2.5.17.1 near_point.c
(./simnet/release/src/vehicle/libsrc/libnear/near_point.c)

Includes:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"
"p_sim.h"
"librva_util.h"
"libnear.h"

2.5.17.1.1 near_get_next_veh_near_point

This routine returns the next vehicle from the specified RVA list (*veh_list_id*), counting from *index*, which is within range (*dist_2*) of the given point (*point*). *Index* is modified.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_list_id	int	Standard
point	VECTOR	/simnet/common/include/global/sim_types.h
dist_2	REAL	/simnet/common/include/global/sim_types.h
index	pointer to int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
vehicles	pointer to pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
num_of_veh	int	Standard
delta	VECTOR	/simnet/common/include/global/sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
vehicles[*index]	pointer to VehicleAppearanceVariant	the next vehicle from the RVA list within the given range
0	pointer to VehicleAppearanceVariant	either no vehicles are on the RVA output list or no vehicle is within the given range

Calls	
Function	**Where Described**
rva_get_output_list	Section 2.5.13.1.2
vec_sub	Section 2.6.2.65.1
vec_dot_prod	Section 2.6.2.54.1

Table 2.5-334: near_get_next_veh_near_point Information.

2.5.17.1.2 near_get_veh_if_still_near_point

This routine returns the given vehicle (*vehicle_id*) if it is within range (*dist_2*) of the specified point (*point*).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
vehicle_id	pointer to VehicleID	basic.h
point	VECTOR	/simnet/common/include/global/sim_types.h
dist_2	REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
vehicle	VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
delta	VECTOR	/simnet/common/include/global/sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
NULL	pointer to VehicleAppearanceVariant	the vehicle is not within range of the given point
vehicle	pointer to VehicleAppearanceVariant	the vehicle is within range of the given point

Calls	
Function	**Where Described**
rva_util_get_veh_app_pkt	Section 2.5.13.1.3
vec_sub	Section 2.6.2.65.1
vec_dot_prod	Section 2.6.2.54.1

Table 2.5-335: near_get_veh_if_still_near_point Information.

2.5.17.1.3 near_get_veh_closest_to_point

This routine finds the vehicle on the RVA ouput list (*veh_list_id*), starting at *index*, which is closest to the given point (*point*).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_list_id	int	Standard
point	VECTOR	/simnet/common/include/global/sim_types.h
dist_2	REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
result	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
index	int	Standard
vehicles	pointer to pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
num_of_veh	int	Standard
delta	VECTOR	/simnet/common/include/global/sim_types.h
current_dist	REAL	/simnet/common/include/global/sim_types.h
min_dist	REAL	/simnet/common/include/global/sim_types.h
min_index	int	Standard

Return Values		
Return Value	**Type**	**Meaning**
NULL	pointer to VehicleAppearanceVariant	no vehicles on the RVA output list
vehicles[min_index]	pointer to VehicleAppearanceVariant	the closest vehicle to the given point

Calls	
Function	**Where Described**
rva_get_output_list	Section 2.5.13.1.2
vec_sub	Section 2.6.2.65.1
vec_dot_prod	Section 2.6.2.54.1

Table 2.5-336: near_get_veh_closest_to_point Information.

2.5.17.1.4 near_get_preferred_veh_near_point

Given that we have a vehicle (*veh_id*) that was near the point during the last frame, this routine tests to see if this vehicle is still close to the given point by calling the routine **near_get_veh_if_still_near_point**(). If the vehicle is still within the given range of the point, that vehicle is returned. If the vehicle is out of range, the routine calls **near_get_veh_closest_to_point**() to get the current closest vehicle off the RVA output list.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
veh_list_id	int	Standard
point	VECTOR	/simnet/common/include/global/sim_types.h
dist_2	REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
result	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h

Return Values		
Return Value	**Type**	**Meaning**
result	pointer to VehicleAppearanceVariant	either *result* = the vehicle that was closest to the point last frame, or *result* = the current closest vehicle to the point

Calls	
Function	**Where Described**
near_get_veh_if_still_near_point	Section 2.5.17.1.2
near_get_veh_closest_to_point	Section 2.5.17.1.3

Table 2.5-337: near_get_preferred_veh_near_point Information.

2.5.17.2 near_vector.c

(./simnet/release/src/vehicle/libsrc/libnear/near_vector.c)

Includes:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"
"p_sim.h"
"librva_util.h"
"libnear.h"

2.5.17.2.1 near_get_next_veh_near_vector

This routine returns the next vehicle from the specified RVA list (*veh_list_id*) whose vector is within angular range (*cos_2*) of the given vector (*vec*).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_list_id	int	Standard
loc	VECTOR	/simnet/common/include/global/sim_types.h
vec	VECTOR	/simnet/common/include/global/sim_types.h
cos_2	REAL	/simnet/common/include/global/sim_types.h
index	pointer to int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
vehicles	pointer to pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
num_of_veh	int	Standard
delta	VECTOR	/simnet/common/include/global/sim_types.h
d_prod	REAL	/simnet/common/include/global/sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
0	pointer to VehicleAppearanceVariant	the next vehicle from the RVA output list within the given range
vehicles[*index]	pointer to VehicleAppearanceVariant	either no vehicles are on the RVA output list or no vehicle is within the given range

Calls	
Function	**Where Described**
rva_get_output_list	Section 2.5.13.1.2
vec_sub	Section 2.6.2.65.1
vec_dot_prod	Section 2.6.2.54.1

Table 2.5-338: near_get_next_veh_near_vector Information.

2.5.17.2.2 near_get_veh_if_still_near_vector

This routine returns the given vehicle (*veh_id*) if its vector is within angular range (*cos_2*) of the specified vector (*vec*).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
loc	VECTOR	/simnet/common/include/global/sim_types.h
vec	VECTOR	/simnet/common/include/global/sim_types.h
cos_2	REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
vehicle	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
delta	VECTOR	/simnet/common/include/global/sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
0	pointer to VehicleAppearanceVariant	the vehicle vector is not within range of the given vector
vehicle	pointer to VehicleAppearanceVariant	the vehicle vector is within range of the given vector

Calls	
Function	**Where Described**
rva_util_get_veh_app_pkt	Section 2.5.13.1.3
vec_sub	Section 2.6.2.65.1
vec_dot_prod	Section 2.6.2.54.1

Table 2.5-339: near_get_veh_if_still_near_vector Information.

2.5.17.2.3 near_get_veh_closest_to_vector

This routine finds the vehicle on the RVA ouput list (*veh_list_id*) whose vector is closest to the given vector (*vec*).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_list_id	int	Standard
loc	VECTOR	/simnet/common/include/global/sim_types.h
vec	VECTOR	/simnet/common/include/global/sim_types.h
cos_2	REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
vehicles	pointer to pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
num_of_veh	int	Standard
delta	VECTOR	/simnet/common/include/global/sim_types.h
d_prod	REAL	/simnet/common/include/global/sim_types.h
current_cos	REAL	/simnet/common/include/global/sim_types.h
min_cos	REAL	/simnet/common/include/global/sim_types.h
min_index	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
0	pointer to VehicleAppearanceVariant	no vehicles on the RVA output list
vehicles[min_index]	pointer to VehicleAppearanceVariant	the closest vehicle to the given vector

Calls	
Function	**Where Described**
rva_get_output_list	Section 2.5.13.1.2
vec_sub	Section 2.6.2.65.1
vec_dot_prod	Section 2.6.2.54.1

Table 2.5-340: near_get_veh_closest_to_vector Information.

2.5.17.2.4 near_get_preferred_veh_near_vector

Given that we have a vehicle (*veh_id*) whose vector was near the given vector during the last frame, this routine tests to see if this vehicle is still near o the given vector by calling the routine **near_get_veh_if_still_near_vector()**. If the vehicle's vector is still within the given angular range of the given vector, that vehicle is returned. If the vehicle is out of range, the routine calls **near_get_veh_closest_to_vector()** to get the current closest vehicle off the RVA output list.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
veh_list_id	int	Standard
loc	VECTOR	/simnet/common/include/global/sim_types.h
vec	VECTOR	/simnet/common/include/global/sim_types.h
cos_2	REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
result	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h

Return Values		
Return Value	**Type**	**Meaning**
result	pointer to VehicleAppearanceVariant	either *result* = the vehicle that was closest to the vector last frame, or *result* = the current closest vehicle to the vector

Calls	
Function	**Where Described**
near_get_veh_if_still_near_vector	Section 2.5.17.2.2
near_get_veh_closest_to-vector	Section 2.5.17.2.3

Table 2.5-341: near_get_preferred_veh_near_vector Information.

2.5.18 librotate
(./simnet/release/src/vehicle/libsrc/librotate [librotate])

This library contains functions related to vehicle internal kinematics.

2.5.18.1 librot_loc.h

Defines:

IKIN_SET
IKIN_OLD
IKIN_NEW

IKIN_CLASS_PLAIN
IKIN_CLASS_STAB_PARENT
IKIN_CLASS_STAB_CHILD
IKIN_CLASS_STAB_ORPHAN
IKIN_CLASS_OFFSET

IKIN_COMMAND_FREEZE
IKIN_COMMAND_ANGLE
IKIN_COMMAND_RATE
IKIN_COMMAND_ANGLE_AND_RATE
IKIN_COMMAND_VECTOR_STAB
IKIN_COMMAND_POINT_STAB
IKIN_COMMAND_RATE_STAB
IKIN_COMMAND_ORIENT
IKIN_COMMAND_LOCATE

External Procedure Declarations:

rotate_relate_init()
rotate_relate_simul()
rotate_set_transform()
rotate_set_location()
rotate_break_links()
rotate_valid_angle()

2.5.18.2 librotate.h

Includes "sim_types.h"

Defines:
- ROTATE_MAX_CHILDREN
- ROTATE_NULL
- ROTATE_ELEMENT_DEF()

The *rotate_element* structure is composed of the following members:

1) parent
2) child[ROTATE_MAX_CHILDREN]
3) class
4) rotate_node
5) axis
6) angle
7) sin_ang
8) cos_ang
9) rate
10) transform_index
11) location_index
12) stop_neg
13) stop_pos
14) max_rate
15) dynamics_on
16) dynamic_gain
17) dynamic_zero
18) pre_command_function()
19) post_command_function()
20) the *orientation* structure member, which is composed of:
    a) command
    b) matrix
    c) rate
    d) angle
    e) last_angle
    f) angle_status
    g) node
21) the *location* structure member, which is composed of:
    a) command
    b) vector
    c) rate
    d) node
22) the *stab_info* union member, which is composed of:
    a) stab_vector
    b) the *family* structure, which is composed of the following members:
        1) stab_child
        2) priority_child
23) stab_base
24) stab_cross_prod
25) stab_dot_prod
26) stab_command_status

The type ROTATE_ELEMENT is defined as a rotate_element structure.

External Declarations:

```
rotate_init()
rotate_simul()
world()
rotate_init_cig_element()
rotate_reassign_cig_element()
rotate_reset_cig_list()
rotate_get_cig_info()
rotate_send_msgs()
hull()
rotate_hull_init()
rotate_hull_simul()
rotate_allocate_element()
rotate_init_element()
rotate_init_stab_family()
rotate_init_stab_orphan()
rotate_init_offset_element()
rotate_prioritize_elements()
rotate_set_child_priority()
rotate_set_stops()
rotate_set_max_rate()
rotate_set_dynamic_characteristics()
rotate_set_dynamic_state()
rotate_set_no_rotate()
rotate_set_mat()
rotate_set_angle()
rotate_set_rate()
rotate_set_angle_and_rate()
rotate_set_current_angle()
rotate_modify_stab_offset()
rotate_set_stab_vector()
rotate_set_stab_vector_in_coordinates()
rotate_set_stab_current_position()
rotate_set_stab_current_position_in_coordinates()
rotate_set_stab_point()
rotate_set_stab_point_in_coordinates()
rotate_set_stab_rate()
rotate_set_stab_rate_in_coordinates()
rotate_set_loc()
rotate_get_angle()
rotate_get_sin_angle()
rotate_get_cos_angle()
rotate_get_rate()
rotate_get_mat()
rotate_get_loc()
```

2.5.18.3 rot_comm.c

The routines in this file are called by other modules in the vehicle software package, and not by other routines within librotate. If none of the routines are called outside librotate, this file will not be linked with the rest of the vehicles code, and none of the included libraries need be linked.

This file contains routines which allow graphical displays to be driven in a transparent fashion. Nodes in the CIG configuration tree can be associated with rotate elements and messages will be sent to the CIG to update these nodes. Note that the hull node, which is a child of the world node, is maintained here.

"stdio.h"
"sim_dfns.h"
"sim_types.h"
"libhull.h"
"libkin.h"
"libmsg.h"
"librotate.h"

Declaring the **malloc**() gives the ability to dynamically allocate memory.

A cig_element is defined and established for each CIG node which is to be driven by the rotate package. All the elements are stored in a linked list. The cig_element structure is defined with the following members:

next	-- The next element on the list.
cig_id	-- The id of the CIG which paints the diplay for this node.
cig_node	-- The number of this node.
parent	-- The parent of htis node, that is, the parent define in the CIG configuration tree. This parent node does not have to be the parent of the child defined in the rotate tree.
child	-- The child node.

In addition, the following variables are declared:

first_element	-- A pointer to the first element in the list. Initially, the list is empty.
current_element	-- A pointer to the element whose information will be sent to the CIG next

The hull is defined with ROTATE_ELEMENT_DEF(hull_element)

2.5.18.3.1 rotate_init_cig_element

This routine properly stores CIG node information in the local list. If an element has already been defined for this *cig_id - cig_node* pair, the *parent_and_child_information* is updated, otherwise a new element is added to the list. Parameters are represented as follows:

cig_id -- The id of the CIG which displays this node.
cig_node -- The number of the node as defined in the CIG configuration file.
parent -- A pointer to the rotate element which is the parent of the node as defined in the CIG configuration file.
child -- A pointer to the rotate element which the CIG node is attached to.

First the *current_element* variable is set to point at the beginning of the list of CIG elements. A search through the list is performed to locate the *cig_id - cig_node* pair. If this pair has been assigned an element before, its parent and child information is updated, and *current_element* is reset to point at the beginning of the list. If the pair has not been assigned an element, memory is allocated. If memory has been exhausted, an error message is printed. Once the element is assigned, it is inserted at the beginning of the list and the parent and child information is filled in.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
cig_id	int	Standard
cig_node	int	Standard
parent	pointer to ROTATE_ELEMENT	Section 2.5.28.2 librotate.h
child	pointer to ROTATE_ELEMENT	Section 2.5.28.2 librotate.h

Errors	
Error	**Reason for Error**
LIBROTATE(rotate_init_cig_element): FATAL	cannot allocate entry for CIG node

Table 2.5-342: rotate_init_cig_element Information.

2.5.18.3.2 rotate_reassign_cig_element

This routine allows branches of the CIG configuration tree to be attached to different parts of the simulated vehicle. For example, if a vehicle has a turret, a gun which can elevate is attached, as is a sensor that can traverse and elevate with respect to the turret. One CIG node could be attached to the turret and another, initially, to the gun. By changing the child in the CIG element originally assigned to the turret, the node could be attached to the sensor.

The routine returns a pointer to the rotate element which was the child associated with the CIG element. Parameters and variables are represented as follows:

cig_id -- The id of the CIG which displays this node
cig_node -- The number of the node as defined in the CIG configuration file. The combination of *cig_id* and *cig_node* uniquely identifies a single CIG display node or CIG element.
child -- A pointer to the rotate element which will now be the child associated with the CIG element.
element -- A pointer to an element in the list.
old_child -- A pointer to the rotate element last assigned to this CIG element.

This is accomplished by setting *element* to point at the beginning of the list and searching for the CIG element. If the CIG element is found, it resets the child and returns the pointer to the old child. If the CIG element is not found, it returns a NULL pointer.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
cig_id	int	Standard
cig_node	int	Standard
child	pointer to ROTATE_ELEMENT	Section 2.5.28.2 librotate.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
element	pointer to struct cig_element	Section 2.5.18.3 rot_comm.c
old_child	pointer to ROTATE_ELEMENT	Section 2.5.28.2 librotate.h
Return Values		
Return Value	**Type**	**Meaning**
ROTATE_NULL	pointer to ROTATE_ELEMENT	the CIG element was not found
old_child	pointer to ROTATE_ELEMENT	the old child associated with the CIG element

Table 2.5-343: rotate_reassign_cig_element Information.

2.5.18.3.3 rotate_reset_cig_list

This routine resets *current_element* to point to the beginning of the list.

2.5.18.3.4 rotate_get_cig_info

This routine passes back the information the CIG needs about the element pointed at by *current_element*. If valid information is returned, TRUE is returned and *current_element* is set to point at the next entry in the list. If the end of the list has been reached, FALSE is returned and *current_element* is reset to point at the beginning of the list.

cig_id -- A pointer to the id of the CIG associated with the current CIG element.
cig_node -- A pointer to the number of the node associated with the current CIG element.
mat -- A pointer to the matrix which contains the transformation from the parent in the current CIG element to the child.
loc -- A pointer to the location of the child in the current CIG element with respect to the the parent expressed in the parent's coordinate system.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
cig_id	pointer to int	Standard
cig_node	pointer to int	Standard
mat	pointer to T_MAT_PTR	/simnet/common/include/global/sim_types.h
loc	pointer to pointer to REAL	/simnet/common/include/global/sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	valid information about the element is returned to the CIG
TRUE	int	the end of the list has been reached

Calls	
Function	**Where Described**
rotate_get_mat	Section 2.5.18.5.5
rotate_get_loc	Section 2.5.18.5.12

Table 2.5-344: rotate_get_cig_info Information.

2.5.18.3.5 rotate_send_msgs

This routine sends messages to the CIG with information about the CIG elements on the list. The messages start with the element currently pointed at by *current_element* and continue through to the end of the list. The only time *current_element* should be pointing anywhere except at the front of the list is during the execution of this routine. In this (the standard) case, this routine will send messages for all elements on the list. This routine should be called by the user when it is appropriate to send messages to the CIG. Variables are represented by the following:

buf_mask -- The CIG id.
cig_node -- The CIG node.
orientation -- The orientation of the node.
location -- The location of the node.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
buf_mask	int	Standard
cig_node	int	Standard
orientation	T_MAT_PTR	/simnet/common/include/global/sim_types.h
location	pointer to REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
rotate_get_cig_info	*Section 2.5.18.3.4*
multi_cig_msg_prepend_rts4x3_matrix	Section 2.1.2.2.2.82.1

Table 2.5-345: rotate_send_msgs Information.

2.5.18.3.6 hull

This routine returns the pointer to the hull. Rotate elements are generally maintained by the vehicles code module responsible for them. The exceptions are the world and the hull, which are maintained here (in librotate). In later versions of the code, the hull element will be moved to its own module.

Return Values		
Return Value	**Type**	**Meaning**
&hull	pointer to ROTATE_ELEMENT	the pointer to the hull element

Table 2.5-346: hull Information.

2.5.18.3.7 rotate_hull_init

This routine initializes the hull rotate element. It should be called by the user before **rotate_init()** is called. It is assumed that there is a CIG configuration node (1) that is associated with the hull. The rotate and CIG elements are initialized, and the initial orientation and location of the hull are set.

Calls	
Function	**Where Described**
rotate_init_element	Section 2.5.18.4.2
world	Section 2.5.18.6.8
rotate_init_cig_element	Section 2.5.18.3.1
rotate_set_mat	Section 2.5.18.4.16
kinematics_get_w_to_h	Section 2.5.8.2.1
rotate_set_loc	Section 2.5.18.4.30
kinematics_get_o_to_h	Section 2.5.8.2.4

Table 2.5-347: rotate_hull_init Information.

2.5.18.3.8 rotate_hull_simul

This routine updates the current orientation and position of the hull. It should be called every tick after the hull kinematics have been performed and before **rotate_simul()** is called.

Calls	
Function	**Where Described**
rotate_set_mat	Section 2.5.18.4.16
kinematics_get_w_to_h	Section 2.5.8.2.1
rotate_set_loc	Section 2.5.18.4.30
kinematics_get_o_to_h	Section 2.5.8.2.4

Table 2.5-348: rotate_hull_simul Information.

2.5.18.4 rot_element.c

This file contains the routines that operate on specific rotate elements. An element is a node in a tree structure that represents and tracks a particular coordinate system. The routines in this file provide the means to initialize elements, change their characteristics, give commands for rotation and translation, and obtain information about them.

Includes:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"libmatrix.h"
"librotate.h"
"librot_loc.h"

Procedure Declarations:
malloc()
rotate_init_stab_element()

2.5.18.4.1 rotate_allocate_element

This routine dynamically allocates memory for an element, initializes the essential entries in it, and returns a pointer to it. This is an alternative to the more standard method of statically allocating in vehicle specific code using the ROTATE_ELEMENT_DEF macro. Using the dynamic method requires the user to be very careful about the order in which nodes are allocated and initialized: an element must be allocated before it is declared as a parent of any other element. This declaration must take place when the child node is initialized. Variables are represented as follows:

i -- A counter.
new_element -- The pointer to the new element.

The pointers to the element's parent and children are initialized to NULL pointers and the pointer to the element is returned. If the element's allocation fails, an error message is printed and a NULL pointer is returned.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
new_element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
Return Values		
Return Value	**Type**	**Meaning**
ROTATE_NULL	pointer to ROTATE_ELEMENT	the element's allocation failed
new_element	pointer to ROTATE_ELEMENT	a pointer to the newly allocated element

Table 2.5-349: rotate_allocate_element Information.

2.5.18.4.2 rotate_init_element

This routine performs all the tasks required to initialize an element and insert it into the rotate tree structure. This initialization must be performed for all elements. This routine must be called before any other action is taken on the element and before **rotate_init**() is called. The order in which the parent element and its children are initialized is not important. After being initialized, this element is of IKIN_CLASS_PLAIN. Other initialization routines must be called to modify its class. Parameters are represented as follows:

element	-- A pointer to the rotate element to be initialized. Note that the element must be declared statically or dynamically by the module responsible for it, generally the vehicle specific code. This declaration must be performed using the macro ROTATE_ELEMENT_DEF which is defined in "librotate.h" or by using the routine **rotate_allocate_element**().
parent	-- The pointer to the element which is the parent of this element.
axis_x	-- It can be shown that the relationship between any two cooridnate systems can be expressed as an axis and angle of rotation. When the angle is 0.0, the systems are coincident. For any angle, the axis has the same expression in either system. This parameter is the X coordinate of the axis of rotation.
axis_y	-- The Y coordinate of the axis of rotation.
axis_z	-- The Z coordinate of the axis of rotation.
angle	-- The initial angle of rotation in radians.
stop_neg	-- The angle in radians that the element is not allowed to pass if it is rotating in the negative direction. If this is not a valid angle (>(-PI) and <=PI), the element does not have a negative stop.
stop_pos	-- The angle in radians that the element is not allowed to pass if it is rotating in the positive direction. If this is not a valid angle (>(-PI) and <=PI), the element does not have a positive stop.
max_rate	-- The maximum rate the element is allowed to turn in radians per tick.
loc_x	-- The X coordinate of the location of the element with respect to its parent in the parent's coordinate system in meters.
loc_y	-- The Y coordinate of the location in meters.
loc_z	-- The Z coordinate of the location in meters.

In order to initialize the element, the routine first checks to see if the element is the world rotate element. If the element is the world, then the element has no parent, since the world is the top of the rotate tree. If the element is not the world, then the routine makes the element a child of its parent. If, however, this element has already been initialized, it is already a child of its parent. If the element can not make itself a child of its parent, the initialization fails. Since all elements are declared using ROTATE_DEF_ELEMENT or **rotate_allocate_element**(), the pointers to a parent element's children are NULL until a child is adopted; a parent will know nothing of its children until the child attaches itself. In order to attach itself, the pointer to the child element is stored in the first NULL entry in the parent's child list.

Once attached, the entry is initialized. First, the entries of the rotate element are filled in, and the axis vector is normalized. The dynamics are initially turned off and the gain and zero values are initialized such that the element would have an instantaneous response if the dynamics were turned on without setting them. The command pointers are initialized to NULL. The node orientation is initially locked in place. If the element is the world, its location cannot be moved. If the element is not the world, a command is issued to place the element; the placement will occur during the initial pass through **rotate_exec()**. If the initialization was successful, the routine returns TRUE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
parent	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
axis_x	REAL	/simnet/common/include/global/sim_types.h
axis_y	REAL	/simnet/common/include/global/sim_types.h
axis_z	REAL	/simnet/common/include/global/sim_types.h
angle	REAL	/simnet/common/include/global/sim_types.h
stop_neg	REAL	/simnet/common/include/global/sim_types.h
stop_pos	REAL	/simnet/common/include/global/sim_types.h
max_rate	REAL	/simnet/common/include/global/sim_types.h
loc_x	REAL	/simnet/common/include/global/sim_types.h
loc_y	REAL	/simnet/common/include/global/sim_types.h
loc_z	REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard

Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	procedure failed
TRUE	int	the initialization was successful

Calls	
Function	**Where Described**
world	Section 2.5.18.6.8
vec_normalize	Section 2.6.2.63.1
rotate_valid_angle	Section 2.5.18.6.7
rotate_set_stops	Section 2.5.18.4.9
rotate_set_max_rate	Section 2.5.18.4.10

Table 2.5-350: rotate_init_element Information.

2.5.18.4.3 rotate_init_stab_family

The goal of stabilization is to align the base vector, which is fixed in the child's coordinate system, with the stab vector,which is fixed in some other coordinate system. In the case of a stab family, two coordinate systems can be rotated to accomplish this goal. This routine performs the initializations required to configure the pair to a stab family. Paramenters are represented as follows:

stab_child -- A pointer to the child of the stab family pair.
base_x -- The base vector is fixed to the child's system and is aligned with the stab vector. This parameter is the X coordinate of the base vector expressed in the child's coordinates.
base_y -- The Y coordinate of the base vector.
base_z -- The Z coordinate of the base vector.
priority_child -- TRUE if the rotations are done to minimize the rotation of the child or FALSE if the rotation of the parent is minimized.

The routine first initializes the child and set its class, and then initializes the parent and sets its class. The base vector of the parent is the child's axis. The two systems are then connected by setting *stab_child* and setting the priority.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
base_x	REAL	/simnet/common/include/global/sim_types.h
base_y	REAL	/simnet/common/include/global/sim_types.h
base_z	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
rotate_init_stab_element	Section 2.5.18.4.5

Table 2.5-351: rotate_init_stab_family Information.

2.5.18.4.4 rotate_init_stab_orphan

element -- A pointer to the element to be stabilized.
base_x -- The base vector is fixed to the child's system and is aligned with the stab vector. This parameter is the X coordinate of the base vector expressed in the child's coordinates.
base_y -- The Y coordinate of the base vector.
base_z -- The Z coordinate of the base vector.

In order to stabilize a stab orphan, only one coordinate system can be rotated to align the base vector with the stab vector. This routine performs the initializations required to configure the element to a stab orphan.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
base_x	REAL	/simnet/common/include/global/sim_types.h
base_y	REAL	/simnet/common/include/global/sim_types.h
base_z	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
rotate_init_stab_element	Section 2.5.18.4.5

Table 2.5-352: rotate_init_stab_orphan Information.

2.5.18.4.5 rotate_init_stab_element

This routine fills in the entries in the rotate element structure that deal with stabilization. Note that in a stab parent, the child's axis acts like a base vector. This is shown elsewhere. First the base vector is filled in and normalized. The cross and dot products of the base vector and the axis are found. The command status in initialized. Parameters are represented as follows:

element -- A pointer to the element to be stabilized.
base_x -- The base vector is fixed to the child's system and is aligned with the stab vector. This parameter is the X coordinate of the base vector expressed in the child's coordinates.
base_y -- The Y coordinate of the base vector.
base_z -- The Z coordinate of the base vector.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
base_x	REAL	/simnet/common/include/global/sim_types.h
base_y	REAL	/simnet/common/include/global/sim_types.h
base_z	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
vec_normalize	Section 2.6.2.63.1
vec_cross_prod	Section 2.6.2.66.1
vec_dot_prod	Section 2.6.2.54.1

Table 2.5-353: rotate_init_stab_element Information.

2.5.18.4.6 rotate_init_offset_element

An offset element is one that is aligned with its parent but is offset. This routine initializes this type of element. The parameter *element* is a pointer to the element which is to be in the offset class.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Table 2.5-354: rotate_init_offset_element Information.

2.5.18.4.7 rotate_prioritize_elements

Sometimes it is desirable to have one node processed before another. As an example, it may be necessary to stabilize a low priority node to a point expressed in the high priority node's coordinates. The routine returns TRUE if it is possible to reorder the children of an element such that *high* is operated on before *low*, and returns FALSE otherwise. Parameters are represented as follows:

high -- A pointer to the element which is to be operated on first.
low -- A pointer to the element which is to be operated on last.

The routine first checks to see if *high* is an ancestor of *low*. If so, *high* will be operated on first, and TRUE is returned. If not, the routine tries to find the common ancestor of *high* and *low*. The variable *current_high* is first set to *high* and is then set to each of its ancestors until the top of the tree is reached. Next, the variable *current_low* is set to *low* and is then set to each of its ancestors until the top of the tree is reached. A check is made to see if the parent of *current_low* and the parent of *current_high* are the same. If so, the common ancestor has been found. If *current_high* and *current_low* are the same, and since the possiblility of *high* being an ancestor of *low* has already been eliminated, *low* is either the same as *high* or it is a direct ancestor; FALSE is returned since *low* must be operated on before *high*.

Next, each of the children of the common ancestor is looped through. If *current_high* is reached first, it is being operated on first, and TRUE is returned. If *current_low* is reached first, the two elements are swapped in the array of children.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
high	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
low	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
current_high	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
current_low	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
parent	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
i	int	Standard
index	int	Standard

Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	it is not possible to reorder the children
TRUE	int	it is possible to reorder the children

Table 2.5-355: rotate_prioritize_elements Information.

2.5.18.4.8 rotate_set_child_priority

This routine sets the child priority flag for a stab family. TRUE is returned if rotation of the child is minimized, and FALSE is returned if rotation of the parent is minimized. Parameters are represented as follows:

element -- A pointer to the element of interest.
priority -- The child priority flag.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
priority	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	the rotation of the child is minimized
FALSE	int	the rotation of the parent is minimized

Errors	
Error	**Reason for Error**
ROTATE: rotate_set_child_priority	trying to prioritize inappropriate class

Table 2.5-356: rotate_set_child_priority Information.

2.5.18.4.9 rotate_set_stops

element -- A pointer to the element of interest.
stop_neg -- The angle in radians that the element cannot turn past when rotating in the negative direction.
stop_pos -- The angle in radians that the element cannot turn past when rotating in the positive direction.

This routine sets the stops associated with an element. If the stop setting is an invalid angle (<= -PI or > PI), then there is no stop in that direction. An angle indicating no negative stop should be less than -PI and an angle indicating no positive stop should be greater than PI in order for the stop determining algorithm to perform efficiently. This routine accepts -PI but converts it to the valid value of PI. If the stops are set at the same angle, they must be separated so the stop algorithm will work properly. This separation will allow the element to rotate virtually through the whole circle but it will stop at the stop point when going in either direction.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
stop_neg	REAL	/simnet/common/include/global/sim_types.h
stop_pos	REAL	/simnet/common/include/global/sim_types.h

Table 2.5-357: rotate_set_stops Information.

2.5.18.4.10 rotate_set_max_rate

This routine sets the maximum allowed rate of rotation. Parameters are represented as follows:

element -- A pointer to the element of interest.
max_rate -- The maximum allowed rate of rotation of the element in either direction in radians per tick.

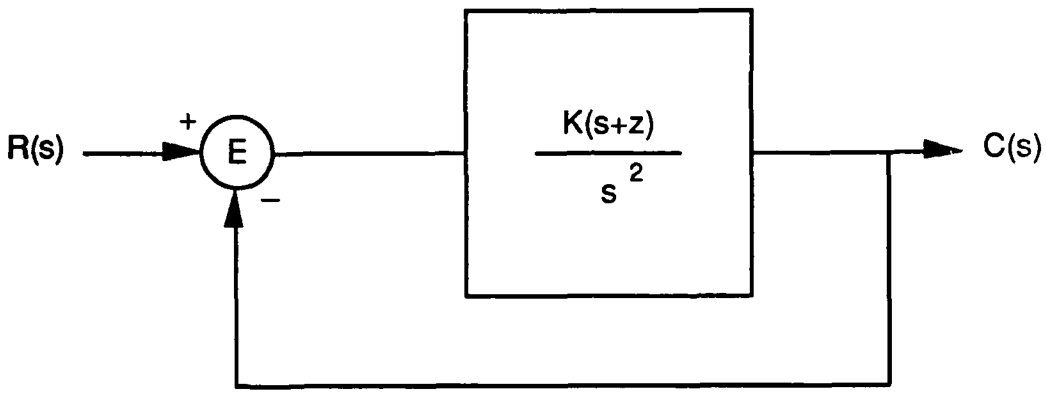
Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
max_rate	REAL	/simnet/common/include/global/sim_types.h

Table 2.5-358: rotate_set_max_rate Information.

2.5.18.4.11 rotate_set_dynamic_characteristics

element -- A pointer to the element of interest.
nat_freq -- The natural frequency of the system in radians per tick.
damping_fac -- The damping factor of the system.

Nodes can be given dynamic characteristics in rotation. A very simple transfer function is used to represent the passive dynamics of the mass, stiffness, and damping and its modification by active control. The criteria for such behavior is that the system should have zero steady-state error for constant velocity input and should be stable. This behavior is exhibited by the system illustrated below. This system has unity and negative feedback. The two pure integrators in the open loop transfer function result in zero steady-state error. The single pole makes the system stable.



The resultant transfer function of this system is:

$$T(s) = \frac{C(s)}{R(s)} = \frac{K(s+z)}{s^2 + Ks + Kz}$$

The response characteristics of this system can be determined from the denominator of this equation (the characteristic equation). The standard form of the second order characteristic equation is:

$$s^2 + 2dws + w^2$$

where *d* is the damping factor and *w* is the natural frequency. Expressing the characteristic equation of the system derived above in this form yields:

$$K = 2dw$$

$$z = \frac{w}{2d}$$

This routine expresses the node's dynamic characteristics in terms of the natural frequency and damping factor. The dynamic characteristics can be set at any time after the node has been initialized and can be modified at any time. The characteristics should be set before the dynamics of the node are turned on. There is no need to set the characteristics if the dynamics are not to be used for the node.

When the dynamics are turned on, the node reacts to inputs of desired position and desired rate. Each of the command states tries to generate reasonable values for these inputs.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
nat_freq	REAL	/simnet/common/include/global/sim_types.h
damping_fac	REAL	/simnet/common/include/global/sim_types.h

Table 2.5-359: rotate_set_dynamic_characteristics Information.

2.5.18.4.12 rotate_set_dynamic_state

element -- A pointer to the element of interest.
dynamics_on -- The desired state (ON or OFF) of the dynamics of the element.

The dynamics of a node are initially turned off and are active only when turned on. This routine turns the dynamics of a node on or off. The dynamic characteristics of the node should be set before they are turned on.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
dynamics_on	int	Standard

Table 2.5-360: rotate_set_dynamic_state Information.

2.5.18.4.13 rotate_set_pre_command_function

element -- A pointer to the element of interest.
function_ptr -- A pointer to a function that returns an integer and passes a pointer to a rotate element as an argument. This function will be called every tick just before the element is operated on.

This routine allows the user to specify a function which will be called after all higher priority elements have been operated on but before this element is operated on. The function declaration should appear as follows:

```
int function_ptr (element)
ROTATE_ELEMENT *element;
    {
      .
      .
      .
    }
```

The pointer passed in the call will be *element*. The return value is not used and has no meaning. If a function has been set previously and it is desired to not have a function called, this routine is used to set the function pointer to ROTATE_NULL.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
function_ptr	pointer to a function that returns an int	Standard

Table 2.5-361: rotate_set_pre_command_function Information.

2.5.18.4.14 rotate_set_post_command_function

This routine allows the user to specify a function which will be called after all higher priority elements have been operated on and after this element is operated on. The function declaration should look like:

```
int function_ptr (element)
ROTATE_ELEMENT *element;
        {
          .
          .
          .
        }
```

element -- A pointer to the element of interest.
function_ptr -- A pointer to a function that returns an integer and passes a pointer to a rotate element as an argument. This function will be called every tick just after the element is operated on.

The pointer passed in the call will be *element*. The return value is not used and has no meaning. If a function has been set previously and it is desired to not have a function called, this routine is used to set the function pointer to ROTATE_NULL.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
function_ptr	pointer to a function that returns an int	Standard

Table 2.5-362: rotate_set_post_command_function Information.

2.5.18.4.15 rotate_set_no_rotate

This routine issues a command to lock the element in place. The dynamic characteristics are ignored. The parameter *element* represents a pointer to the element of interest.

If the element was previously stabilized and is a member of a stab family, the other member of the family will also be locked into place. Typically, another call would be made to the other member to verify its command status.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Table 2.5-363: rotate_set_no_rotate Information.

2.5.18.4.16 rotate_set_mat

element -- A pointer to the element of interest.
parent_to_self -- A pointer to the parent to self transformation matrix.

This routine issues a command to set the orientation matrix of an element. Note that the element can no longer be thought of in terms of rotating about its axis and that the dynamic characteristics have no bearing.

If the element was previously stabilized and is a member of a stab family, the other member of the family will also be locked into place. Typically, another call would be made to the other member to verify its command status.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
parent_to_self	T_MATRIX	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_copy	Section 2.6.2.39

Table 2.5-364: rotate_set_mat Information.

2.5.18.4.17 rotate_set_angle

element -- A pointer to the element of interest.
angle -- The desired angle with respect to the parent [radians].

This routine issues a command to rotate to a specified angle relative to the parent. If dynamics are turned on, this angle is the desired angle. The transition into this command mode results in the desired rate being set to 0.0. If this routine is called every tick, the difference between successive calls is used as the desired rate. If a tick passes without a call to this routine, the desired rate is set to 0.0. If the element was previously stabilized and is a member of a stab family, the other member of the family will also be locked into place. Typically, another call would be made to the other member to verify its command status.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
angle	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
rotate_valid_angle	Section 2.5.18.6.7

Table 2.5-365: rotate_set_angle Information.

2.5.18.4.18 rotate_set_rate

element -- A pointer to the element of interest.
rate -- The desired rate of rotation in radians per tick.

This routine issues a command to rotate at a specified rate. If dynamics are turned on, the current angle is used as the desired angle.

If the element was previously stabilized and is a member of a stab family, the other member of the family will also be locked into place. Typically, another call would be made to the other member to verify its command status.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
rate	REAL	/simnet/common/include/global/sim_types.h

Table 2.5-366: rotate_set_rate Information.

2.5.18.4.19 rotate_set_angle_and_rate

element -- A pointer to the element of interest.
angle -- The desired angle with respect to the parent [rad].
rate -- The desired rate of rotation in radians per tick.

This routine is used primarily to set the desired angle and rate of a node when dynamics are turned on. If the dynamics are turned off, the node will turn at the specified rate (if possible) until the desired angle is reached.

If the element was previously stabilized and is a member of a stab family, the other member of the family will also be locked into place. Typically, another call would be made to the other member to verify its command status.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
angle	REAL	/simnet/common/include/global/sim_types.h
rate	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
rotate_valid_angle	Section 2.5.18.6.7

Table 2.5-367: rotate_set_angle_and_rate Information.

2.5.18.4.20 rotate_set_current_angle

This routine will cause the node to come to rest at its current position. If dynamics are turned off, this will have the same effect as the routine **rotate_set_no_rotate**() but will be computationally more expensive. If dynamics are turned on, the node will settle onto its current position. The parameter *element* represents a pointer to the element of interest.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Calls	
Function	**Where Described**
rotate_set_angle	Section 2.5.18.4.17

Table 2.5-368: rotate_set_current_angle Information.

2.5.18.4.21 rotate_modify_stab_offset

element -- A pointer to the element of interest.
offset -- Desired angular offset from the stab vector in radians.

This routine modifies the offset used for *element* when it is in stab vector or point stab mode. The node must be a stabilized element to be in stab vector or point stab mode. The offset is reset to zero when the node transitions into either of those command modes or when the coordinate system of the stab vector or stab point changes. If the element is not in vector stab or point stab mode, an error message is printed.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	
offset	REAL	/simnet/common/include/global/sim_types.h

Errors	
Error	**Reason for Error**
ROTATE: rotate_modify_stab_offset	element not in appropriate stab mode

Table 2.5-369: rotate_modify_stab_offset Information.

2.5.18.4.22 rotate_set_stab_vector

element -- A pointer to the element of interest.
stab_vector -- A pointer to the stab vector which is expressed in world coordinates.

This routine sets the stab vector which is expressed in world coordinates by calling the more general routine **rotate_set_stab_vector_in_coordinates**().

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
stab_vector	VECTOR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
rotate_set_stab_vector_in_coordinates	Section 2.5.18.4.23

Table 2.5-370: rotate_set_stab_vector Information.

2.5.18.4.23 rotate_set_stab_vector_in_coordinates

Parameters and variables are represented as follows:

element -- A pointer to the element of interest.
stab_vector -- A pointer to the stab vector which is expressed in the coordinates of node *coords*.
coords -- A pointer to the element whose coordinates the stab vector is expressed in.
parent -- A pointer to the parent in the family.
child -- A pointer to the child in the family.

This routine sets the stab vector which is in the coordinates of element *coords*. *coords* should be moved before *element* by calling **rotate_prioritize_elements**() except when this routine is called from **rotate_set_stab_vector**(), since the world is always fixed.

This routine sets the command mode of *element*, and, if it is in a stab family, sets its other family member. If *element* is not in a stab family or is not a stab orphan, this routine prints an error message and returns with no action.

When in the stab vector command mode, an attempt is made to align the base vector with the stab vector then rotate each node away from the stab vector by some offset angle. The offset for all affected nodes is set to zero when this command mode is entered from some other command mode or if *coords* changes. The offset for an element can be set using the routine **rotate_modify_stab_offset**().

When dynamics are turned on, the desired angle is the angle which would align the base vector with the stab vector plus the offset. The difference between consecutive desired angles is the desired rate. When this command mode is entered from another mode or when the value of *coords* changes, the desired rate will be zero.

If this routine is called every tick, the desired rate will be as described above. If it is called while in this command mode but it was not called during the previous tick, the desired rate will be reset to zero. The assumption is that the stab vector could be changing continously and that this routine would be called every tick to reflect this change. The difference between consecutive desired angles would be an appropriate measure of desired rate in this case.

If this routine is not called every tick, it is assumed that the stab vector is not changing continously. When the routine is called, the new stab vector is not assumed to be associated with the previous stab vector and the desired rate is set to zero.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
stab_vector	VECTOR	/simnet/common/include/global/sim_types.h
coords	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
parent	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
child	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Errors	
Error	**Reason for Error**
ROTATE: rotate_set_stab_vector	trying to stabilize inappropriate class

Calls	
Function	**Where Described**
vec_normalize	Section 2.6.2.63.1

Table 2.5-371: rotate_set_stab_vector_in_coordinates Information.

2.5.18.4.24 rotate_set_stab_current_position

This routine finds the current position of the base vector in world coordinates and uses it to set the stab vector.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Calls	
Function	**Where Described**
rotate_set_stab_current_position_in_ coordinates	Section 2.5.18.4.25

Table 2.5-372: rotate_set_stab_current_position Information.

2.5.18.4.25 rotate_set_stab_current_position_in_coordinates

element -- A pointer to the element of interest.
coords -- A pointer to the element in whose coordinates the base vector is to be expressed.

This routine finds the current position of the base vector in coordinates of *coords* and uses it to set the stab vector. If the element is not in a stab class, an error message is printed.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
coords	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
child	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
stab_vector	VECTOR	/simnet/common/include/global/sim_types.h

Errors	
Error	**Reason for Error**
ROTATE: rotate_set_current_position_stab	trying to stabilize inappropriate class

Calls	
Function	**Where Described**
vec_mat_mul	Section 2.6.2.56.1
rotate_get_mat	Section 2.5.18.5.5
rotate_set_stab_vector_in_coordinates	Section 2.5.18.4.23

Table 2.5-373: rotate_set_stab_current_position_in_coordintates Information.

2.5.18.4.26 rotate_set_stab_point

element -- A pointer to the element of interest.
stab_point -- A pointer to a vector which stores the location at which the base vector should point. The point is expressed in meters in world coordinates.

This routine sets the stab point which is expressed in world coordinates. It calls the more general routine **rotate_set_stab_point_in_coordinates()** to set the vector.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
stab_point	VECTOR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
rotate_set_stab_point_in_coordinates	Section 2.5.18.4.27
world	Section 2.5.18.6.8

Table 2.5-374: rotate_set_stab_point Information.

2.5.18.4.27 rotate_set_stab_point_in_coordinates

element -- A pointer to the element of interest.
stab_point -- A pointer to a vector which stores the location at which the base vector should point. The point is expressed in meters in the coordinates of *coords*.
coords -- A pointer to the element in whose coordinates the stab point is expressed.

This routine sets the point to which the base vector of a family or orphan is to point. This point is in the coordinates of element *coords*. *coords* should be moved before *element* by calling **rotate_prioritize_elements()** except when this routine is called from **rotate_set_stab_point()**, since the world is always fixed.

This routine sets the command mode of *element*, and, if it is in a stab family, sets its other family member. If *element* is not in a stab family or is not a stab orphan, this routine prints an error message and returns with no action.

When in the stab point command mode, an attempt is made to point the base vector at the stab point then rotate each node away from the stab point by some offset angle. The offset for all affected nodes is set to zero when this command mode is entered from some other command mode or if *coords* changes. The offset for an element can be set using the routine **rotate_modify_stab_offset()**.

When dynamics are turned on, the desired angle is the angle which would point the base vector at the stab point plus the offset. The differance between consecutive desired angles is the desired rate. When this command mode is entered from another mode or when the value of *coords* changes, the desired rate will be zero.

If this routine is called every tick, the desired rate will be as described above. If it is called while in this command mode but it was not called during the previous tick, the desired rate will be reset to zero. The assumption is that the stab point could be changing continously and that this routine would be called every tick to reflect this change. The differance between consecutive desired angles would be an appropriate measure of desired rate in this case.

If this routine is not called every tick, it is assumed that the stab point is not changing continously. When it is called, the new stab point is assumed to be unassociated with the previous stab point and the desired rate is set to zero.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
stab_point	VECTOR	/simnet/common/include/global/sim_types.h
coords	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
parent	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
child	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Errors	
Error	**Reason for Error**
ROTATE: rotate_set_stab_point	trying to stabilize inappropriate class

Calls	
Function	**Where Described**
vec_copy	Section 2.6.2.59.1

Table 2.5-375: rotate_set_stab_point_in_coordinates Information.

2.5.18.4.28 rotate_set_stab_rate

element -- A pointer to the element of interest.
rate -- The desired rate of rotation of the node in radians per tick.

This routine causes the node to stabilize in the world and turn at the specified rate by calling the more general routine **rotate_set_stab_rate_in_coordinates**().

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
rate	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
world	Section 2.5.18.6.8
rotate_set_stab_rate_in_coordinates	Section 2.5.18.4.29

Table 2.5-376: rotate_set_stab_rate Information.

2.5.18.4.29 rotate_set_stab_rate_in_coordinates

element -- A pointer to the element of interest.
rate -- The desired rate of rotation of the node in radians per tick.
coords -- A pointer to the element in whose coordinates the element is stabilized.

This routine causes the stab family or orphan to stabilize in the desired coordinates and turn at the desired rate. When this command mode is entered from another command mode or when the coordinate system changes, the current location of the base vector in the coordinates of *coords* is determined and saved as the stab vector. *coords* should be moved before *element* by calling **rotate_prioritize_elements**() except when this routine is called from **rotate_set_stab_rate**(), since the world is always fixed.

If *element* is not in a stab family, this routine prints an error message and returns with no action.

When the nodes are operated on, an attempt is made to align the base vector with the stab vector to point the base vector in its original direction. Each node in the family is then turned by the amount specified in *rate*. After the stab orphan or both members of the stab family have been moved, the position of the base vector in *coords* is determined and saved and the process repeats.

When this command mode is entered from another mode or when a new coordinate system is specified, both members of a stab family enter this mode. The specified rate is set for *element*, and the rate for the other member of the family is set to zero. Usually, this routine would be called for both members of a stab family. The first call would perform all the necessary initialization and set the rate of the node specified in the call. The second call would simply set the rate for the other family member. Any call to this routine which does not cause a transition in command mode or does not specify new coordinates simply sets a new rate for the specified node.

The action of this command mode when dynamics are turned on is very similar to the other stabilized modes. The desired angle is the angle which will align the base vector with the stab vector plus the specified rate. The desired rate is not *rate*, but is the difference between consecutive desired angles. When there is a transition into this command mode or when new coordinates are specified, the desired rate is set to zero. Subsequent calls to this routine or their frequency do not affect the desired rate.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
rate	REAL	/simnet/common/include/global/sim_types.h
coords	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
parent	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
child	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Errors	
Error	**Reason for Error**
rotate_set_stab_rate_in_coordinates	trying to stabilize inappropriate class

Calls	
Function	**Where Described**
vec_mat_mul	Section 2.6.2.56.1
rotate_get_mat	Section 2.5.18.5.5

Table 2.5-377: rotate_set_stab_rate_in_coordinates Information.

2.5.18.4.30 rotate_set_loc

element -- A pointer to the element of interest.
location -- The location of the element expressed in meters in its parent's coordinates.
This routine issues a command to set the location of the element.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
location	VECTOR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
vec_copy	Section 2.6.2.59.1

Table 2.5-378: rotate_set_loc Information.

2.5.18.4.31 rotate_get_angle

This routine returns the current angle (in radians) of *element* with respect to its parent. *element* represents a pointer to the element of interest.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
Return Values		
Return Value	**Type**	**Meaning**
element->angle	REAL	the current angle of *element*

Table 2.5-379: rotate_get_angle Information.

2.5.18.4.32 rotate_get_sin_angle

This routine returns the sine of the current angle of *element* with respect to its parent. *element* represents a pointer to the element of interest.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
Return Values		
Return Value	**Type**	**Meaning**
element->sin_ang	REAL	the sine of the current angle of *element*

Table 2.5-380: rotate_get_sin_angle Information.

2.5.18.4.33 rotate_get_cos_angle

This routine returns the cosine of the current angle of *element* with respect to its parent. *element* represents a pointer to the element of interest.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
Return Values		
Return Value	**Type**	**Meaning**
element->cos_ang	REAL	the cosine of the current angle of *element*

Table 2.5-381: rotate_get_cos_angle Information.

2.5.18.4.34 rotate_get_rate

This routine returns the current rate of rotation of *element* with respect to its parent. *element* represents a pointer to the element of interest.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
Return Values		
Return Value	**Type**	**Meaning**
element->rate	REAL	the current rate of rotation of *element*

Table 2.5-382: rotate_get_rate Information.

2.5.18.5 rot_relate.c

This file contains routines that determine and maintain the relationships between coordinate systems; these routines find and store the transformation matrices, location vectors, etc. The coordinate systems are one of the characteristics of the rotation elements. The elements are assigned positions within a hierarchal tree structure. Connected pairs in the tree have a parent-child relationship. All but one of the elements has a single parent and may have one or more children. The top element in the tree is always the world and has no parent.

This file contains the transformation element definition. An array of transformation elements is declared with an entry for each node pair. An element contains the information needed to determine and maintain the transformation between the nodes. Some of the members of this structure make use of the fact that each of the nodes are assigned a number with the number of a parent always being lower than the number of any of its children. The transform_element structure is defined with the following members:

exists	-- TRUE if the transformation from one of the nodes to the other is known or can be obtained directly from rotation element information, and FALSE otherwise.
value	-- The value of this transformation. The value is based on the number and rate of requests for the transformation from one of the nodes to the other.
requests	-- The number of requests made during the current time step.
product	-- If the nodes associated with this element are not a parent and child, this transformation can be found by multiplying two other matrices (which may or may not currently exist) together. For example, if this element is associated with the transformation betweens nodes 4 and 7, then the transformation can be found by multiplying the transformations between nodes 4 and 2 and nodes 2 and 7. The number of the optimal intermediate node (2 in this example) is stored here.
test	-- The number of a potential intermediate node.
link	-- If the associated nodes are a parent and child, *link* is a pointer to the child. If not, *link* is a null pointer. If this is a parent-child pair (not null), this transformation always exists.
matrix_exists	-- TRUE if the transformation from the lower numbered node to the higher has been determined, and FALSE otherwise.
matrix	-- The transformation matrix from the lower numbered coordinate system to the higher.
transpose_exists	-- TRUE if the transformation from higher numbered node to the lower has been determined, and FALSE otherwise.
transpose	-- The transformation matrix from the higher numbered coordinate system to the lower.

This file contains the location element definition. An element contains, or has the information necessary to determine, the location of one node with respect to another. The location is expressed in the originating node's coordinate system. For example, the location of 7 with respect to 4 is the vector from 4 to 7 expressed in 4's coordinate system. An array of these elements is declared with an entry for each directed node pair; for example, there is an entry for 7 with respect to 4 and another entry for 4 with respect to 7. The location_element structure is defined with the following members:

exists	-- TRUE if the location is known, and FALSE otherwise.
value	-- The value of this location. The value is based on the number and frequency of requests made.
requests	-- The number of requests for this location made during the current time step.
inverse	-- The index of the inverse location. If, for example, this location is 7 with respect to 4, this is index of the location element of 4 with respect to 7.
matrix	-- The index into the transformation array of the transformation between the nodes associated with this element.
vector	-- The location vector.

This file contains the path search tree_element definition. These types of elements are used in finding the shortest and most valuable path from one node to another. All the potential paths are stored in a tree structure and the tree is built one generation at a time. The tree_element structure is defined with the following members:

parent	-- The element immediately preceding this element in the potential path.
cousin	-- The next element in this generation of potential path nodes.
index	-- The number of the node associated with this element. Used to index into the node status array.

This file contains the break element definition. These types of elements are used to build lists of transformation or location elements that are invalidated when the position or orientation of a rotation node changes. The lists are linked lists. The break_element structure is defined with the following members:

next	-- The next element in the list.
index	-- The index to the element in the transformation or location array which is invalidated.

VALUE_DECAY_FACTOR is defined as the percentage of previous value added to the current number of requests to determine the new value.

Declarations:

node_count	-- The number of nodes.
node_count_minus_1	-- The number of nodes minus 1.
transform_count	-- The number of transform elements.
location_count	-- The number of location elements.
num_tree_nodes	-- The number of reserved tree elements.

To allow for any number of nodes, much of the needed memory is dynamically allocated. The following pointers to this memory are set to ROTATE_NULL at compile time to show that no memory is associated with them.

node_status	-- An indication of the availability of a node in a search path is stored in an array pointed to by *node_status*. There is one entry for each node. If the entry is TRUE, the node is available, and FALSE if not.
mul_path	-- The order of transformation multiplications is stored as an ordered list of nodes in *mul_path*. The longest possible path would include each node once.

transform_list is declared as the array of transform elements.

location_list is declared as the array of location elements.

tree_list is declared as the array of tree elements.

node_transform_break is declared as an array of pointers to break elements. The array has an entry for each node and each entry points to the beginning of a linked list of break elements. Each element in a list indicates a transformation that is invalidated when the orientation of the associated node is changed.

node_location_break is declared as a similar array of pointers. The linked lists associated with each node indicate the locations that are invalidated when the node changes its orientation or position with respect to its parent.

node_location_break_ends is declared as a similar array of pointers. The linked lists associated with each node indicate the locations that are invalidated when the node changes its position with respect to its parent but not when the orientation is changed.

The following static functions are declared:

rotate_number_node()
rotate_fill_permanent_tree()
rotate_find_transform_path()
rotate_path_val()
rotate_save_path()
rotate_mat()
rotate_transform_index()
rotate_location_index()
dump_transform()
dump_location()
dump_break_list()

2.5.18.5.1 rotate_relate_init

This routine performs the initialization tasks related to the tracking of the relationships between nodes. In order to handle rotation trees of any size, much of the memory used is allocated dynamically. This routine manages this memory and initializes a number of variables. Internal variables are represented as follows:

i	-- A counter.
child_node	-- The child's node number in a parent-child pair.
from	-- The first node in a path.
to	-- The last node in a path.
transform_index	-- An index into the transform list.
location_index	-- An index into the location list.
current_break	-- A pointer to a break element.

The pointers to the blocks of memory that are dynamically allocated are initialized to ROTATE_NULL in the compiler. If the simulation has been previously initialized during the execution of the simulation software, the pointers are no longer ROTATE_NULL but point to the memory last allocated. Any previously allocated memory is freed.

Three groups of break elements exist. Within each group, a linked list of break elements is associated with each node. For each group, this memory is freed by following each list to free each break element and then freeing the array of pointers for the group.

The node count to is set to zero, and the nodes are numbered. The world will be node zero and each parent will have a number lower than any of its children. When the numbering of nodes is completed, *node_count* will contain the total number of nodes. This allows *node_count_minus_1* to be found.

The number of combinations of k items out of n items is n!/(k!(n-k)!). In this case, we are interested in the number of node pairs, so k = 2. The number of pairs then becomes n(n-1)/2. *transform_count* is set to this value and *transform_count* + 1 transform elements are allocated. The extra element is used to represent the transformation of any node to itself.

Each ordered pair has a location element, therefore there are twice as many location elements as transform elements. *location_count* is set and *location_count* + 1 location elements are allocated. The extra element is used to represent the location of a node with respect to itself. *node_status* and *mul_path* are allocated. *num_tree_nodes* is found and *tree_list* is allocated. *node_transform_break*, *node_location_break* and *node_location_break_ends* are allocated

Each entry of each of the break arrays is filled with ROTATE_NULL. This indicates that no linked lists have been formed yet.

The initialization for each node pair is first performed. The indices into *transform_list* and *location_list* are found and each of the elements in the entries are initialized. The location list entry for the inverse location is initialized. The extra entries in *transform_list* and *location_list*, for transformation from a node to itself, are initialized. Note that the *location_list* extra entry is its own inverse and that it points to the extra entry in the *transform_list*.

The identity matrix is stored as the transformation matrix. Even though the matrix has been filled in, the existence flag for this transformation is still set FALSE.

A zero length vector is stored as the location vector Even though the vector has been filled in, the existence flag for this location is still set FALSE.

The entries in the transform and location lists that represent links between parents and children in the rotate node tree are found. These entries permanently exist and that existence is flagged

Break lists are built by looping through every node pair.

The index into the transform list is found and tested for existence of the transform. If it exists (that is, if the two nodes are a parent-child pair) the next node pair is tested.

The path between the nodes is found. The only transform elements that currently exist are those that represent parent-child pairs, the transforms that permanently exist. This, along with the fact that no loops exist in the node tree, means that there is only one path between the nodes and that it only contains transformations between parents and children. The path (which is stored in *mul_path* as a list of nodes) is followed from the beginning to the end.

Each consecutive pair of nodes in *mul_path* is a parent child pair. The larger node number is the child. Since each child has only one parent, changing the transformation or location between a parent and child can be viewed as changing the orientation or position of the child. The child node is found and incremented *i*. The while loop construction allows each node pair to be processed.

If the child's orientation is changed, the transformation between the *from* and *to* nodes (if it exists) is invalidated. A break element is constructed for the *from-to* transformation and placed at the beginning of the transform break list for the child node.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
child_node	int	Standard
from	int	Standard
to	*int*	Standard
transform_index	int	Standard
location_index	int	Standard
current_break	pointer to the structure break_element	Section 2.5.18.5 rot_relate.c

Errors	
Error	**Reason for Error**
LIBROTATE(rotate_relate_init): FATAL	cannot allocate: transform_list location_list node_status mul_path tree_list node_transform_break node_location_break node_location_break_ends

Calls	
Function	**Where Described**
rotate_number_node	Section 2.5.18.5.2
world	Section 2.5.18.6.8
rotate_transform_index	Section 2.5.18.5.11
rotate_location_index	Section 2.5.18.5.14
mat_ident	Section 2.6.2.43.1
vec_init	Section 2.6.2.61.1

rotate_fill_permanent_tree	Section 2.5.18.5.3
rotate_find_transform_path	Section 2.5.18.5.6

Table 2.5-383: rotate_relate_init Information.

2.5.18.5.2 rotate_number_node

This routine assigns a number to the node specified in the rotation element, *element*. A pointer to the world node is passed into the first call to this routine. It then numbers each of the nodes in the tree recursively. Because of the order in which functions are performed, the number of a parent is always less than the number of any of its children. After all the nodes have been numbered, *node_count* contains the total number of nodes.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard

Calls	
Function	**Where Described**
rotate_number_node	Section 2.5.18.5.3

Table 2.5-384: rotate_number_node Information.

2.5.18.5.3 rotate_fill_permanent_tree

This routine sets the existence flag of the transform between the element and its parent and the location of the element with respect to its parent. It also fills in the information linking the transform and location elements to the rotate element. This routine calls itself recursively after a pointer to the world is passed in the first call. No assignments are made when *element* represents the world.

The index into the transform list for the transform between this element and its parent is found and stored in the element structure. The existence of this transform is indicated and a pointer to the rotate element is stored.

The index into the location list for the location of the element with respect to its parent is found and saved in the element structure. The existence of this location is indicated. Note that the location of the element with respect to its parent permanently exists, however, the location of the parent with respect to the element does not. The location of the element with respect to its parent may only be changed explicitly; the values in the location vector change although it remains a valid location. If the element changes its orientation with respect to its parent, the location of the parent with respect to the element is no longer valid and must be determined.

The existence of the links to all of the element's children are then established.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
index	int	Standard
Calls		
Function	**Where Described**	
rotate_transform_index	Section 2.5.18.5.11	
rotate_location_index	Section 2.5.18.5.14	
rotate_fill_permanent_tree	Section 2.5.18.5.3	

Table 2.5-385: rotate_fill_permanent_tree Information.

2.5.18.5.4 rotate_relate_simul

This routine updates the values of the transform and location elements. This routine is the only librotate routine which requires a knowledge of the passage of time.

The transform list is updated by finding a new value from the sum of the number of request made during the current tick and a fraction of the current value. The request count is then zeroed. If the number of requests made per tick is constant, then the new value of the transform will be less than {requests / (1 - VALUE_DECAY_FACTOR)}; if the number of requests made in a tick is bounded, then the value is bounded.

Internal Variables		
Variable	Type	Where Typedef Declared
i	int	Standard

Table 2.5-386: rotate_relate_simul Information.

2.5.18.5.5 rotate_get_mat

This routine finds a transformation matrix between coordinate systems. If this matrix is pre-multiplied by a vector expressed in *from* coordinates, the result is the same vector expressed in *to* coordinates. The routine returns a pointer to the desired transformation matrix. Parameters and variables are represented as follows:

from -- A pointer to a rotate element from whose coordinate system a transformation is desired.
to -- A pointer to a rotate element to whose coordiante system a transformation is desired.
index -- The index of the transform element.

First the routine calculates *index*. Note that if *index* is equal to *transform_count*, a transformation from a node to itself has been requested. In this case the identity matrix is returned. Otherwise, the transformation matrix is determined and the requests counter is incremented.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
from	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
to	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
index	int	Standard

Return Values		
Return Value	**Type**	**Meaning**
transform_list[index].matrix	T_MAT_PTR	a pointer to the identity matrix
rotate_mat (from->rotate_node, to-->rotate_node)	T_MAT_PTR	a pointer to the transformation matrix

Calls	
Function	**Where Described**
rotate_transform_index	Section 2.5.18.5.11
rotate_find_transform_path	Section 2.5.18.5.6

Table 2.5-387: rotate_get_mat Information.

2.5.18.5.6 rotate_find_transform_path

This routine determines the path on the tree between the given nodes *from* and *to*. Parameters are represented as follows:

from -- A pointer to the node number from which the path starts.
to -- A pointer to the node number to which the path ends.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
from	int	Standard
to	int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
j	int	Standard
tree_index	int	Standard
depth	int	Standard
path_not_found	int	Standard
val	REAL	/simnet/common/include/global/sim_types.h
path_val	REAL	/simnet/common/include/global/sim_types.h
generation	pointer to structure tree_element	Section 2.5.18.5 rot_relate.c
search	pointer to structure tree_element	Section 2.5.18.5 rot_relate.c
temp_element	pointer to structure tree_element	Section 2.5.18.5 rot_relate.c

Errors	
Error	**Reason for Error**
LIBROTATE(rotate_find_transform_path): FATAL	Exhausted tree elements
LIBROTATE(rotate_find_transform_path): FATAL	Exhausted connections

Calls	
Function	**Where Described**
rotate_path_val	Section 2.5.18.5.7
rotate_save_path	Section 2.5.18.5.8

Table 2.5-388: rotate_find_transform_path Information.

2.5.18.5.7 rotate_path_val

This routine returns the path value of a path of a specified length. Parameters are represented as follows:
path -- the path of interest.
length -- the length of *path*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
path	an array of int	Standard
length	int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
index	int	Standard
current_val	REAL	/simnet/common/include/global/sim_types.h
new_val	REAL	/simnet/common/include/global/sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
0.0	REAL	no path exists
current_val + transform_list[index].value	REAL	the value of the specified path

Calls	
Function	**Where Described**
rotate_path_val	Section 2.5.18.5.7
rotate_transform_index	Section 2.5.18.5.11

Table 2.5-389: rotate_path_val Information.

2.5.18.5.8 rotate_save_path

This routine saves the tree path between the node numbered *from* to the node numbered *to*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
from	int	Standard
to	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
index	int	Standard
Calls		
Function	**Where Described**	
rotate_transform_index	Section 2.5.18.5.11	
rotate_save_path	Section 2.5.18.5.8	

Table 2.5-390: rotate_save_path Information.

2.5.18.5.9 rotate_mat

This routine finds a transformation matrix between coordinate systems. If this matrix is pre-multiplied by a vector expressed in *from* coordinates, the result is the same vector expressed in *to* coordinates.

from -- The number of the node from whose coordinate system a transformation is desired.
to -- The number of the node to whose coordiante system a transformation is desired.
index -- The index of the transform element associated with this node pair.
link -- A pointer to the child's rotate element for a parent-child pair.

The routine first determines *index*. The routine then determines if the transformation exists. A transformation exists if the transformation or its transpose has been found, or if the transformation is associated with a parent-child pair and can be generated directly. There is one transformation for each node pair.

First the routine determines if the transformation exists. There is one transformation element for each node pair. If the value of *from* is less than *to*, then the matrix associated with this transformation is desired. If the matrix does not exist, it is generated either from the transpose or from a defined parent-child link. If the value of *from* is greater than *to*, the transpose of the transformation matrix is desired. If this transpose does not exist, it must be generated from the matrix. If the matrix does not exist, the transpose can be generated from a defined parent-child link . Once the matrix is found, its existence is flagged and a pointer to the transpose is returned. If neither the transformation or the matrix can be found, an error is indicated. If the value of *from* equals *to*, a request for the transformation of a node to itself has been made. This is an error condition. The error is indicated and the identity matrix is returned.

If the transformation does not exist, it must be generated by matrix multiplication. If *from* is less than *to*, the matrix is generated by multiplying the transformation from the *from* node to the product node by the transformation from the product node to the *to* node. The existence of the transformation and the matrix are flagged and a pointer to the matrix is returned. If *from* is greater than *to*, the transpose is desired. The transpose is generated by multiplying the transformation from the *from* node to the product node by the transformation from the product node to the *to* node. The existence of the transformation and the matrix are flagged and a pointer to the transpose is returned. If the value of *from* equals *to*, a request for the transformation of a node to itself has been made. This is an error condition. The error is indicated and the identity matrix is returned.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
from	int	Standard
to	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
index	int	Standard
link	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
Return Values		
Return Value	**Type**	**Meaning**
transform_list[index].index	T_MAT_PTR	the transformation matrix
transform_list[index].transpose	T_MAT_PTR	the transformation transpose
transform_list[index].matrix	T_MAT_PTR	the transformation matrix
Errors		
Error	**Reason for Error**	
LIBROTATE(rot_mat):	FATAL - Cannot form matrix FATAL - Cannot form transpose WARNING - Dummy matrix exists? WARNING - Requesting transform from node to itself	
Calls		
Function	**Where Described**	
mat_transpose	Section 2.6.2.51.1	
mat_form	Section 2.6.2.42.1	
mat_mat_mul	Section 2.6.2.32.1	
rotate_mat	Section 2.5.18.5.9	
rotate_transform_index	Section 2.5.18.5.11	

Table 2.5-391: rotate_mat Information.

2.5.18.5.10 rotate_set_transform

This routine is called when the transformation between a parent and child is set directly. It is called from **rotate_exec()** in "rot_util.c". It takes *element*, a pointer to a rotate element, as a parameter.

The matrix is copied from the rotate element into the transform element, and the flag is set.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Calls	
Function	**Where Described**
mat_copy	Section 2.6.2.39.1

Table 2.5-392: rotate_set_transform Information.

2.5.18.5.11 rotate_transform_index

This routine returns the index into *transform_list* for a node pair. Parameters are represented as follows:

from -- The number of the first node of the pair.
to -- The number of the second node of the pair.

One transfromation element exists for each node pair, i.e. the transformation from node 4 to node 7 uses the same element as the transformation from node 7 to node 4. One element is used for all transformations from any node to itself. The memory for the elements is allocated at initalization. This routine provides a way to access a transformation element given the node pair.

Consider the case where there are N nodes numbered 0 to N-1. All possible transformations can be represented as elements of an array N by N. The rows represent the node the transformation is from and the columns represent the node the transformation is to. The elements below the diagonal are transposes of the elements above the diagonal, so memory is not allocated for them. The elements on the diagonal represent a transformation from a node to itself. The last transformation element on the list is allocated for all of these transformations. The remaining elements are then stored row-wise. The element represented by row 0 column 1 has an index of 0 in the transformation list, row 0 column 2 has index 1, continuing to row 0 column (N-1) which has index N-2. The next row follows, i.e. row 1 column 2 has index N-1.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
from	int	Standard
to	int	Standard

Return Values		
Return Value	**Type**	**Meaning**
to * node_count + to - (from + 1) * (from + 2) / 2	int	the transform_list index if a matrix pair is given
to * node_count + from - (to + 1) * (to + 2) / 2	int	the transform_list index if a transpose pair is given
transform_count	int	the index of the identity element

Table 2.5-393: rotate_transform_index Information.

2.5.18.5.12 rotate_get_loc

This routine finds the location of the origin of the *to_element* coordinate system relative to the origin of the *from_element* coordinate system expressed in *from_element* coordinates. Parameters and variables are represented as follows:

from_element	--	A pointer to the rotate element which is the reference coordinate system for this request.
to_element	--	A pointer to the rotate element whose location is requested.
from	--	The node number of *from_element*.
to	--	The node number of *to_element*.
test_from	--	An intermediate *from* node number
test_to	--	An intermediate *to* node number.
main_index	--	The index into the location list of the *from* to *to* node pair.
index	--	The index into the location list of the *test_from* to *to* node pair.
last_index	--	The value of *index* for the previous pass through a loop.
temp_index	--	The index into the location list of the *test_from* to *test_to* node pair.
i	--	A counter.
j	--	A counter.
length	--	The number of elements in the path from *from* to *to* minus 1.
temp_vector	--	A vector used to store intermediate results.

First, *from* and *to*, and *main_index* are found. If the request is for the location of a node with respect to itself or for a location that has already been determined, the pointer to the location vector is returned.

The inverse of the requested vector is the location of the *from* coordinate system relative to the *to* coordinate system expressed in *to* coordinates. The requested location is then the negative of the inverse times the transformation from *to* coordinates to *from* coordinates.

length is found. The path length for any location that permanently exists and its inverse is 0. All other path lengths must be greater or equal to 3. If the path length of the requested location is less than 3 (i.e. *length* is less than 2), an error indication is made.

Consider the following case. The location of node 4 with respect to node 0 is requested; *from* is 0 and *to* is 4. The path between them is {0, 1, 2, 3, 4}. The path length is 5 and *length* is equal to 4. The definition of a path essentially states that the location or inverse of the location of any two adjacent nodes in the path exists as does the transformation between any two adjacent nodes. Assuming that no other locations are known, the transformation between two nodes will be expressed as a_C_b, for example, the transformation from 4 to 3 is 4_C_3. The location is expressed as a_v_b, for example, the location of 3 with respect to 4, expressed in 4's coordinates is 4_v_3. The math used to find the inverse described above would then be 3_v_4 = -4_v_3 * 4_C_3. For the example request 0_v_4, the following steps would need to be performed:

> 3_v_4 permanently exists.
> 2_v_4 = 3_v_4 * 3_C_2 + 2_v_3
> 2_v_4 = 2_v_4 * 2_C_1 + 1_v_2
> 1_v_4 = 1_v_4 * 1_C_0 + 0_v_1

Note that the intermediate results are generated by working backwards through the path. Using the example, at this point it is known that 0_v_4 (or 4_v_0 and 4_C_0) does not exist. The first intermediate result that exists is found by looping forward through the path, in the example, check to see if 1_v_4 or 2_v_4 or 3_v_4 exists.

Even if the desired intermediate result may not exist, its inverse and appropriate transformation may. If this is the case, the intermediate location can be found. For example, 2_v_4 does not exist but 4_v_2 and 4_C_2 do. 2_v_4 = 4_v_2 * 4_C_2.

If the location of the next adjacent node pair exists, the next intermediate result can be found. For example, 2_v_4 was found in the last pass. 1_v_2 exists. The next intermediate result is 1_v_4 = 2_v_4 * 2_C_1 + 1_v_2. If the location of the next adjacent node pair does not exist, its inverse does. Find the next intermediate result. For example, 2_v_4 was found in the last pass. 1_v_2 does not exist but 2_v_1 does. The next intermediate result is 1_v_4 = (2_v_4 - 2_v_1) * 2_C_1.

The last intermediate result found is actually the desired location and a pointer to it is returned.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
from_element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
to_element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
from	int	Standard
to	int	Standard
index	int	Standard
last_index	int	Standard
temp_index	int	Standard
i	int	Standard
tree_index	int	Standard
path_not_found	int	Standard
current_val	REAL	/simnet/common/include/global/sim_types.h
new_val	REAL	/simnet/common/include/global/sim_types.h
temp_vector	VECTOR	/simnet/common/include/global/sim_types.h
generation	struct tree_element	Section 2.5.18.5 rot_relate.c
search	struct tree_element	Section 2.5.18.5 rot_relate.c
temp_element	struct tree_element	Section 2.5.18.5 rot_relate.c
last_element	struct tree_element	Section 2.5.18.5 rot_relate.c

Return Values		
Return Value	**Type**	**Meaning**
location_list[location_count].vector	pointer to REAL	/simnet/common/include/global/sim_types.h
location_list[index].vector	pointer to REAL	/simnet/common/include/global/sim_types.h

Errors	
Error	**Reason for Error**
LIBROTATE(rotate_get_loc)	FATAL - Exhausted tree elements FATAL - Exhausted connections
Calls	
Function	**Where Described**
rotate_location_index	Section 2.5.18.5.14
vec_mat_mul	Section 2.6.2.56.1
rotate_mat	Section 2.5.18.5.9
vec_neg	Section 2.6.2.62.1
vec_add	Section 2.6.2.57.1
vec_sub	Section 2.6.2.65.1

Table 2.5-394: rotate_get_loc Information.

2.5.18.5.13 rotate_set_location

This routine is called when the location of a child with respect to its parent is set directly. It is called from **rotate_exec**() in "rot_util.c".

First, copy the location from the rotate element into the location element. If this is a parent-child pair, this location permanently exists and no existence flags need to be set.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Calls	
Function	**Where Described**
vec_copy	Section 2.6.2.59.1

Table 2.5-395: rotate_set_location Information.

2.5.18.5.14 rotate_location_index

This routine returns the index into *location_list* for an ordered node pair. Parameters are represented as follows:

from -- The number of the first node of the pair.
to -- The number of the second node of the pair.

One location element exists for each ordered node pair. This means that the location element for node 4 to node 7 is different than the element for node 7 to node 4, although only one element is used for the location of a node with respect to itself. The memory for the elements is allocated at initialization. This routine provides a way to access a location element given the nodes of interest.

Consider the same array described in the comments of **rotate_transform_index**(). In this case, the elements below the diagonal are of interest. The elements on the diagonal represent the location of a node with respect to itself. One location element is allocated for all these locations and is stored at the end of the list. The remaining elements are stored row-wise. This could be thought of as an array with N-1 rows and N-2 columns. The column number of elements in the upper diagonal of this effective array is one less than the actual column number. The lower diagonal is not affected.

The last element in the list is returned, and represents the location of a node with respect to itself.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
from	int	Standard
to	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
node_count_minus_1 * from + to - 1	int	The last element in the list
node_count_minus_1 * from + to	int	The last element in the list
location_count	int	The last element in the list

Table 2.5-396: rotate_location_index Information.

2.5.18.5.15 rotate_break_links

When the location and/or orientation of an element with respect to its parent is changed, a number of previously calculated transformations and locations are invalidated. This routine performs the invalidation.

element -- A pointer to the rotate element that has been operated on.
translation -- TRUE if the location of *element* with respect to its parent has changed, and FALSE otherwise.
orientation -- TRUE if the orientation of *element* with respect to its parent has changed, and FALSE otherwise.
rotation -- TRUE if the change in orientation is due to a rotation about an axis rather than setting a new orientation directly.
current_break -- A pointer to a break element.
index -- An index into the transformation list.

A location is only invalidated if a change in orientation of location has occurred. Note that since the world is the reference node, an attempt to change its orientation of location is an error. A change in orientation of location invalidates all locations which depend on this element, except for the locations where this element represents the last node pair in the path.

Transformations are invalidated only if a change in orientation occurs. A change of orientation invalidates all transformations which depend on this element. The locations where this element represents the last node pair in the path are invalidated only if a translation has occurred.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
translation	int	Standard
orientation	int	Standard
rotation	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
current_break	pointer to struct break_element	Section 2.5.18.5 rot_relate.c
index	int	Standard

Errors	
Error	**Reason for Error**
LIBROTATE(rotate_break_links): WARNING	Trying to break links due to world movement

Table 2.5-397: rotate_break_links Information.

2.5.18.5.16 dump_transform

This routine is used for debugging purposes only.

Parameters		
Parameter	Type	Where Typedef Declared
from	int	Standard
to	int	Standard

Table 2.5-398: dump_transform Information.

2.5.18.5.17 dump_location

This routine is used for debugging purposes only.

Parameters		
Parameter	Type	Where Typedef Declared
from	int	Standard
to	int	Standard

Calls	
Function	Where Described
rotate_location_index	Section 2.5.18.5.14

Table 2.5-399: dump_location Information.

2.5.18.5.18 dump_break_list

This routine is used for debugging purposes only.

Parameters		
Parameter	Type	Where Typedef Declared
break_list	pointer to struct break_element	Section 2.5.18.5 rot_relate.c

Internal Variables		
Variable	Type	Where Typedef Declared
current_break	pointer to struct break_element	Section 2.5.18.5 rot_relate.c

Table 2.5-400: dump_break_list Information.

2.5.18.5.19 relate_dump_transforms

This routine is used for debugging purposes only.

Internal Variables		
Variable	Type	Where Typedef Declared
from	int	Standard
to	int	Standard

Calls	
Function	Where Described
dump_transform	Section 2.5.18.5.16

Table 2.5-401: relate_dump_transforms Information.

2.5.18.5.20 relate_dump_locations

This routine is used for debugging purposes only.

Internal Variables		
Variable	Type	Where Typedef Declared
from	int	Standard
to	int	Standard

Calls	
Function	Where Described
dump_location	Section 2.5.18.5.17

Table 2.5-402: relate_dump_locations Information.

2.5.18.6 rot_util.c

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libmath.h"
"libmatrix.h"
"librotate.h"
"librot_loc.h"

Procedure Declarations:
ROTATE_ELEMENT_DEF(world_element)
rotate_init_check()
rotate_exec()
rotate_become_legal()
rotate_stab()

2.5.18.6.1 rotate_init

This routine initializes the world. Allocated memory must be initialized for all elements. The routine **rotate_relate_init()** sets up all memory and finds out the relationships between matrices. The routine **rotate_init_check()** does some basic idiot checks, then the routine **rotate_exec()** is called to process the commands.

Calls	
Function	**Where Described**
rotate_init_element	Section 2.5.18.4.2
rotate_relate_init	Section 2.5.18.5.1
rotate_init_check	Section 2.5.18.6.2
rotate_exec	Section 2.5.18.6.4

Table 2.5-403: rotate_init Information.

2.5.18.6.2 rotate_init_check

This routine does some basic idiot checks.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
self	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
state	int	Standard
Return Values		
Return Value	**Type**	**Meaning**

state	int	If state = TRUE, If state = FALSE

Errors	
Error	**Reason for Error**
rotate_init_check:	- Stab child not own child - Stab child not STAB_CHILD class - Parent not STAB_PARENT class - Parents stab child not self

Calls	
Function	**Where Described**
rotate_init_check	Section 2.5.18.6.2

Table 2.5-404: rotate_init_check Information.

2.5.18.6.3 rotate_simul

This routine calls **rotate_relate_simul**() to update information every tick. The routine checks to see that you are not trying to rotate the world.

Errors	
Error	**Reason for Error**
rotate_simul: PANIC!	Trying to rotate world

Calls	
Function	**Where Described**
rotate_relate_simul	Section 2.5.18.5.4
rotate_exec	Section 2.5.18.6.4

Table 2.5-405: rotate_simul Information.

2.5.18.6.4 rotate_exec

This routine is responsible for executing the various IKIN Commands which are listed in "librot_loc.h". The parameter *self* represents a pointer to the element to be operated on.

The routine first calls the **pre_command_function**(), if it is specified.

If the element's location command is IKIN_COMMAND_LOCATE, try to locate the element and then freeze the command until the next update. This is done by the hull every tick.

If the element's orientation command is IKIN_COMMAND_ORIENT, orient the element with the translation matrix returned from **rotate_set_tranform**(). Once oriented, freeze the command until the next update. Note that this command is executed by the hull every tick.

If the element's class is IKIN_CLASS_OFFSET or if its orientation command is IKIN_COMMAND_FREEZE, the rate is set equal to zero, signifying no motion.

If the element is rotating and the orientation command is either IKIN_COMMAND_VECTOR_STAB, IKIN_COMMAND_POINT_STAB, or IKIN_COMMAND_RATE_STAB, determine the angle to move to by calling **rotate_stab**().

If the element is rotating but not stabilizing, and the orientation command is IKIN_COMMAND_ANGLE, first determine whether the angle is inside the stops by calling **rotate_become_legal**(). If the angle is inside the stops, then calculate the fastest rate and correct direction to turn to reach it. If the angle is outside the stops, then the closest angle within the range is found, and the fastest rate and correct direction are calculated.

If the element is rotating but not stabilizing, and the orientation command is IKIN_COMMAND_ANGLE_AND_RATE, determine whether the angle is legal. Turn to the specified angle at the desired rate if the angle is legal. If the angle is not legal, determine a legal angle and the rate to turn. The net rate is the rate to get from where the element should be to where it wants to be.

If the element is rotating but not stabilizing, and the orientation command is IKIN_COMMAND_RATE, determine whether the angle is inside the stops. If the angle is not legal, then the rate is the larger of either the desired rate or the required rate to get to a legal position. If the angle is legal, the rate is the desired rate. Note that a maximum limit can be set for the rate. If the maximum is exceeded, the desired rate will be clipped to the maximum rate. The angle and current rate are updated for the positive and negative directions.

After executing the commands, the routine calls **rotate_break_links**() which records the self, translation, orientation and rotation flags. Any previous calulated matrices that may now be incorrect are invalidated, while matrices that are still correct remain valid.

If the orientation command was IKIN_COMMAND_RATE_STAB and the class is IKIN_CLASS_STAB_PARENT, the stab vector is recorded.

The **post_command_function**() is then called, if it is specified.

Lastly, this routine, **rotate_exec()**, is recursively called for each child, executing the commands for each element in the tree.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
self	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
orientation_command	int	Standard
orientation_angle	REAL	/simnet/common/include/global/sim_types.h
desired_angle	REAL	/simnet/common/include/global/sim_types.h
desired_rate	REAL	/simnet/common/include/global/sim_types.h
net_rate	REAL	/simnet/common/include/global/sim_types.h
temp_ang	REAL	/simnet/common/include/global/sim_types.h
legal_ang	REAL	/simnet/common/include/global/sim_types.h
req_rate	REAL	/simnet/common/include/global/sim_types.h
inside_stops	int	Standard
i	int	Standard
translation	int	Standard
orientation	int	Standard
rotation	int	Standard

Calls	
Function	**Where Described**
rotate_set_location	Section 2.5.18.5.13
rotate_set_transform	Section 2.5.18.5.10
rotate_stab	Section 2.5.18.6.6
rotate_become_legal	Section 2.5.18.6.5
rotate_valid_angle	Section 2.5.18.6.7
real_limit	Section 2.6.1.6.1
rotate_break_links	Section 2.5.18.5.15
vec_mat_mul	Section 2.6.2.56.1
rotate_exec	Section 2.5.18.6.4

Table 2.5-406: rotate_exec Information.

2.5.18.6.5 rotate_become_legal

This routine determines whether the angle is within the positive and negative stops. Parameters are represented as follows:

element -- A pointer to the element of interest.
angle -- The angle to be checked.
new_angle -- A pointer to a legal angle.
rate -- A pointer to the rate in radians per tick to move from *angle* to *new_angle*.

If the angle is inside the stops, the angle is legal and the routine returns TRUE. If the angle is outside the stops, the angle is not legal and the routine returns FALSE. If the angle is outside the stops, a pointer is set to *new_angle*, which is the closest angle within the stops. The rate to turn from *angle* to *new_angle* is calculated.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
angle	REAL	/simnet/common/include/global/sim_types.h
new_angle	pointer to REAL	/simnet/common/include/global/sim_types.h
rate	pointer to REAL	/simnet/common/include/global/sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	The angle is within the stops
FALSE	int	The angle is not within the stops

Table 2.5-407: rotate_become_legal Information.

2.5.18.6.6 rotate_stab

This routine calculates the stabilization angle of the parameter *self*. If *self* is of class stab parent, then *element* is created as the child element.

If stabilizing to a point, the stab vector is calculated. The difference between your location and the point is normalized and placed into your coordinate system. If stabilizing to a vector, the stab vector is placed in your coordinate system. If your class is stab parent, then the child has a base vector which must align to your stab vector.

The routine returns the angle you need to move to for stabilization.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
self	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h
stab_vector	VECTOR	/simnet/common/include/global/sim_types.h
temp_vector	VECTOR	/simnet/common/include/global/sim_types.h
new_angle	REAL	/simnet/common/include/global/sim_types.h
sin_rot	REAL	/simnet/common/include/global/sim_types.h
cos_rot	REAL	/simnet/common/include/global/sim_types.h
stab_dot_axis	REAL	/simnet/common/include/global/sim_types.h
denominator	REAL	/simnet/common/include/global/sim_types.h
temp_1	REAL	/simnet/common/include/global/sim_types.h
temp_2	REAL	/simnet/common/include/global/sim_types.h
diff_angle	REAL	/simnet/common/include/global/sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
self->angle	REAL	the stabilization angle
rotate_valid_angle (self->angle + new_angle)	REAL	the stabilization angle which has been corrected to stay inside the stops

Calls	
Function	**Where Described**
vec_sub	Section 2.6.2.65.1
rotate_get_loc	Section 2.5.18.5.12

vec_normalize	Section 2.3.2.63.1
vec_mat_mul	Section 2.6.2.56.1
rotate_get_mat	Section 2.5.18.5.5
vec_dot_prod	Section 2.6.2.54.1
inv_sin_cos_rad	Section 2.6.1.3.2
vec_cross_prod	Section 2.6.2.66.1
rotate_valid_angle	Section 2.5.18.6.7

Table 2.5-408: rotate_stab Information.

2.5.18.6.7 rotate_valid_angle

This routine determines whether *angle* is within the angle range of -PI to PI and modifies the angle if necessary.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
angle	REAL	/simnet/common/include/global/sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
angle	REAL	a valid angle within -PI to PI

Table 2.5-409: rotate_valid_angle Information.

2.5.18.6.8 world

This routine returns a pointer to the world element.

Return Values		
Return Value	**Type**	**Meaning**
&world_element	pointer to ROTATE_ELEMENT	Section 2.5.18.2 librotate.h

Table 2.5-410: world Information.

2.5.19 libupdate
(./simnet/release/src/vehicle/libsrc/libupdate [libupdate])

This library contains the routines necessary to move the vehicle forward in time.

2.5.19.1 libupdate.c

Includes:
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dyn_state.h"
"kin_state.h"
"dyn_mass.h"

Defines:
CIG_AGL
STARTUP_CEILING
CG_HEIGHT_OFFSET

Declarations:
cig_altitude_above_gnd()
veh_dyn
veh_kin
veh_freeze
freeze_state()
vehicle_place()
init_state
agl
veh_mass
veh_init

2.5.19.1.1 vehicle_update

This routine first calculates the linear forces and torques by calling the dynamics module. Once the acceleration is calculated, the integration is calculated to form the velocity. When the velocity if found, the body can be moved forward in tiem assuming constant acceleration. The new position is calculated and the coordinates are calculated.

Internal Variables		
Variable	Type	Where Typedef Declared
T_sum	VECTOR	/simnet/common/include/global/sim_types.h
R_sum	VECTOR	/simnet/common/include/global/sim_types.h

Calls	
Function	Where Described
dynamics_calc_inertial_forces	Section 2.5.7.1.1
B_w	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h
B_v_cg	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h
vec_add	Section 2.6.2.57.1
dynamics_calc_udot	Section 2.5.7.3.1
B_alpha	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h
vec_mat_mul	Section 2.6.2.56.1
A_a_cg	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h
dynamics_calc_u	Section 2.5.7.2.1
freeze_state	Section 2.5.19.1.28
cig_altitude_above_gnd	Section 2.1.2.2.3.1.1
vehicle_place	Section 2.5.19.1.2
v_mag	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h
mag3	Macro defined in /simnet/common/include/global/sim_macros.h
kinematics_calc_origin_state	Section 2.6.18.1.2
A_w	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h
A_v_origin	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h
A_v_cg	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h
kinematics_form_e	Section 2.6.18.5.1
kinematics_update_p	Section 2.6.18.12.1
kinematics_update_e	Section 2.6.18.11.1
kinematics_form_C	Section 2.6.18.3.1
kinematics_form_G	Section 2.6.18.6.1
kinematics_form_N	Section 2.6.18.4.1
kinematics_form_s	Section 2.6.18.8.1

Table 2.5-411: vehicle_update Information.

2.5.19.1.2 vehicle_place

This routine is called to put the vehicle in a new place. The position is initialized, then the euler parameters are set. The direction and cosine matrices are generated from kinematics.

pos -- position vector to start
fvel -- forward velocity
direction -- direction pointing to start
A_s -- position vector from B* to origin expressed in frame A

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pos	VECTOR	/simnet/common/include/global/sim_types.h
fvel	REAL	/simnet/common/include/global/sim_types.h
direction	REAL	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
A_s	VECTOR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
B_v_cg	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h
v_mag	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h
B_w	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h
kinematics_form_C	Section 2.6.18.3.1
kinematics_form_G	Section 2.6.18.6.1
kinematics_form_N	Section 2.6.18.4.1
kinematics_form_s	Section 2.6.18.8.1

Table 2.5-412: vehicle_place Information.

2.5.19.1.3 vehicle_init

This routine initializes the vehicle position.

pos -- position vector to start
fvel -- forward velocity
direction -- direction pointing to start

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pos	VECTOR	/simnet/common/include/global/sim_types.h
fvel	REAL	/simnet/common/include/global/sim_types.h
direction	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
vehicle_place	Section 2.5.19.1.2

Table 2.5-413: vehicle_init Information.

2.5.19.1.4 vehicle_set_position

This routine sets the vehicle position to *pos*, the position vector.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pos	VECTOR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
kinematics_form_s	Section 2.6.18.8.1

Table 2.5-414: vehicle_set_position Information.

2.5.19.1.4 vehicle_set_orientation

This routine sets the vehicle orientation to *direction*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
direction	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
kinematics_form_C	Section 2.6.18.3.1
kinematics_form_G	Section 2.6.18.6.1
kinematics_form_N	Section 2.6.18.4.1
kinematics_form_s	Section 2.6.18.8.1

Table 2.5-415: vehicle_set_orientation Information.

2.5.19.1.5 kinematics_set_orientation_matrix

This routine forms the vehicle orientation matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
B_C_A	T_MATRIX	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_copy	Section 2.6.2.39.1
mat_transpose	Section 2.6.2.51.1
make_e	Section 2.6.18.9.1
kinematics_form_G	Section 2.6.18.6.1
kinematics_form_N	Section 2.6.18.4.1
kinematics_form_s	Section 2.6.18.8.1

Table 2.5-416: vehicle_set_orientation_matrix Information.

2.5.19.1.6 vehicle_mass_init

This routine initializes the vehicle mass parameters.

Parameters		
Parameter	Type	Where Typedef Declared
mass	REAL	/simnet/common/include/global/sim_types.h
I	T_MATRIX	/simnet/common/include/global/sim_types.h

Calls	
Function	Where Described
dynamics_init	Section 2.5.7.5.1

Table 2.5-417: vehicle_mass_init Information.

2.5.19.1.7 vehicle_restart

This routine restarts the vehicle at its initial position.

Calls	
Function	Where Described
vehicle_place	Section 2.5.19.1.2

Table 2.5-418: vehicle_restart Information.

2.5.19.1.8 vehicle_A_acceleration

This routine returns the acceleration of the vehicle in frame A.

Return Values		
Return Value	Type	Meaning
A_a_cg(veh_dyn)	pointer to REAL	The acceleration of the vehicle in frame A.

Calls	
Function	Where Described
A_a_cg	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h

Table 2.5-419: vehicle_A_acceleration Information.

2.5.19.1.9 vehicle_B_acceleration

This routine returns the acceleration of the vehicle in Frame B.

Return Values		
Return Value	**Type**	**Meaning**
veh_dyn.a	pointer to REAL	the acceleration of the vehicle in frame B.

Table 2.5-420: vehicle_B_acceleration Information.

2.5.19.1.10 vehicle_A_velocity

This routine returns the velocity of the vehicle in Frame A.

Return Values		
Return Value	**Type**	**Meaning**
A_v_cg(veh_dyn)	pointer to REAL	the velocity of the vehicle in frame A.

Calls	
Function	**Where Described**
A_v_cg	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h

Table 2.5-421: vehicle_A_velocity Information.

2.5.19.1.11 vehicle_B_velocity

This routine returns the velocity of the vehicle in Frame B.

Return Values		
Return Value	**Type**	**Meaning**
B_v_cg(ven_dyn)	pointer to REAL	the velocity of the vehicle in frame B.

Calls	
Function	**Where Described**
B_v_cg	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h

Table 2.5-422: vehicle_B_velocity Information.

2.5.19.1.12 vehicle_velocity_magnitude

This routine returns the magnitude of the velocity vector.

Return Values		
Return Value	Type	Meaning
v_mag(veh_dyn)	REAL	The magnitude of the velocity vector.

Calls	
Function	Where Described
v_mag	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h

Table 2.5-423: vehicle_velocity_magnitude Information.

2.5.19.1.13 vehicle_A_r

This routine returns the vehicle's change in position vector in Frame A.

Return Values		
Return Value	Type	Meaning
veh_kin.A_r	pointer to REAL	the change in position vector in Frame A.

Table 2.5-424: vehicle_A_r Information.

2.5.19.1.14 vehicle_angular_velocity

This routine returns the vehicle's angular velocity.

Return Values		
Return Value	Type	Meaning
B_w(veh_dyn)	pointer to REAL	angular velocity of vehicle

Calls	
Function	Where Described
B_w	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h

Table 2.5-425: vehicle_angular_velocity Information.

2.5.19.1.15 vehicle_A_p

This routine returns the vehicle's position vector, from the origin to B* in Frame A.

Return Values		
Return Value	Type	Meaning
veh_kin.A_p	pointer to REAL	vehicle position vector, from origin to B* in Frame A.

Table 2.5-426: vehicle_A_p Information.

2.5.19.1.16 vehicle_B_s

This routine returns the vehicle's position vector, from B* to the origin in Frame B.

Return Values		
Return Value	**Type**	**Meaning**
veh_kin.B_s	pointer to REAL	vehicle position vector, from B* to origin in Frame B.

Table 2.5-427: vehicle_B_s Information.

2.5.19.1.17 vehicle_b2

This routine returns the normal vector of the vehicle in Frame A.

Return Values		
Return Value	**Type**	**Meaning**
veh_kin.A_b2	pointer to REAL	normal vector of vehicle in Frame A.

Table 2.5-428: vehicle_b2 Information.

2.5.19.1.18 vehicle_A_C_B

This routine returns the vehicle's direction cosine matrix from A to B.

Return Values		
Return Value	**Type**	**Meaning**
veh_kin.A_C_B	T_MAT_PTR	vehicle direction cosine matrix from A to B.

Table 2.5-429: vehicle_A_C_B Information.

2.5.19.1.19 vehicle_B_C_A

This routine returns the vehicle's direction cosine matrix from B to A.

Return Values		
Return Value	**Type**	**Meaning**
veh_kin.B_C_A	T_MAT_PTR	vehicle direction cosine matrix from B to A.

Table 2.5-430: vehicle_B_C_A Information.

2.5.19.1.20 vehicle_gravity_vector

This routine returns the vehicle gravity vector in Frame B.

Return Values		
Return Value	Type	Meaning
veh_kin.B_g	pointer to REAL	vehicle gravity vector in frame B.

Table 2.5-431: vehicle_gravity_vector Information.

2.5.19.1.21 vehicle_altitude

This routine returns the altitude of the vehicle.

Return Values		
Return Value	Type	Meaning
veh_kin.A_p[2]	REAL	vehicle altitude

Table 2.5-432: vehicle_altitude Information.

2.5.19.1.22 vehicle_climb_rate

This routine returns the vehicle's climb rate: the velocity of the origin in frame A.

Return Values		
Return Value	Type	Meaning
A_v_origin(veh_dyn)[2]	REAL	vehicle climb rate.

Calls	
Function	Where Described
A_v_origin	Macro defined in /simnet/release/src/libsrc/include/dyn_state.h

Table 2.5-433: vehicle_climb_rate Information.

2.5.19.1.23 vehicle_freeze

This routine sets the *vehicle_freeze* flag to 1. This routine is called to freeze the Stealth vehicle.

2.5.19.1.24 vehicle_thaw

This routine sets the *vehicle_freeze* flag to 0. This routine is called to unfreeze the Stealth vehicle.

2.5.19.1.25 vehicle_freeze_disable

This routine sets the *veh_freeze_enable* flag to 0. This routine is called to disable the freeze capabilities of the vehicle.

2.5.19.1.26 vehicle_torques

This routine sets the vehicle's torques to *val*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	VECTOR	/simnet/common/include/global/sim_types.h

Table 2.5-434: vehicle_torques Information.

2.5.19.1.27 vehicle_forces

This routine sets the vehicle's forces to *val*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	VECTOR	/simnet/common/include/global/sim_types.h

Table 2.5-435: vehicle_forces Information.

2.5.19.1.28 freeze_state

This routine stops the vehicle's motion, setting the velocities and torques to zero.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
w	VECTOR	/simnet/common/include/global/sim_types.h
v	VECTOR	/simnet/common/include/global/sim_types.h

Table 2.5-436: freeze_state Information.

2.5.19.1.29 dump routines

The following routines:

kin_dump()
w_dump()
v_dump()
r_dump()
t_dump()
vehicle_banner()

are used to print information for debugging purposes.

2.5.19.1.30 vehicle_set_init_state

This routine is used by the stealth vehicle to freeze the position of the vehicle when an NLOS missile is flying.

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

Table 2.5-437: vehicle_set_init_state Information.

2.6 Simulation Support Utilities

The software which comprises this CSC is depicted in Figure 2.6-1.

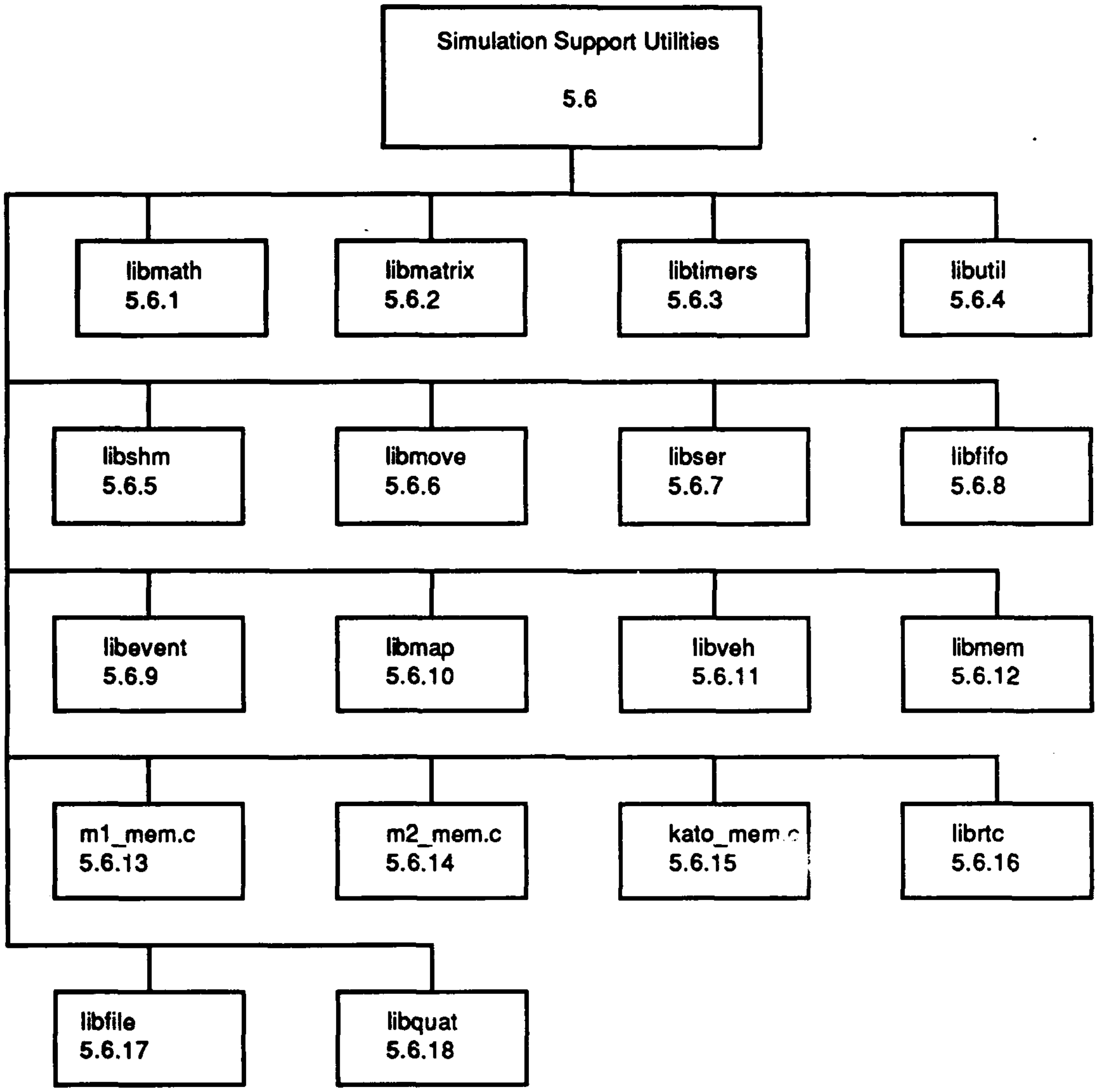


Figure 2.6-1: Simulation Support Software.

The Utilities are a collection of software modules used to perform the generic functions that are not specific to the vehicle simulation. The modules that are included in this category are those whose functionality is general enough to be used (invoked) to perform operations that are not specific to any one subsystem simulated. A primary example is the software library of math routines available to perform the mathematical operations required to implement many of the simulated subsystems. These utilities are represented by the following CSU's:

libmath
libnatrix
libtimers
libutil
libshm
libmove
libser
libfifo
libevent
libmap
libveh
libmem
m1_mem.c
m2_mem.c
kato_mem.c
librtc
libfile
libquat

2.6.1 libmath
(/simnet/release/src/libsrc/libmath [libmath])

Libmath includes routines for the generation of random variables, computation of inverse trig functions, curve fitting and polynomial evaluation.

2.6.1.1 bivar_dist.c
(/simnet/release/src/libsrc/libmath /bivar_dist.c)

This file contains a routine which computes a zero-mean normal distribution along two axes.

Includes:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"

Defines:
K32

2.6.1.1.1 bivariant_normal_distribution

This routine computes a zero-mean normal distribution along two axes. It requires a pointer to an array of floats and a standard deviation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
aptr[]	REAL	sim_types.h
std_dev	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
random_number1	double	Standard
random_number2	double	Standard
theta	double	Standard
magnitude	double	Standard

Table 2.6-1: bivariant_normal_distribution Information.

2.6.1.2 cubic_funct.c
(/simnet/release/src/libsrc/libmath/cubic_funct.c)

This file contains routines which generate and solve a function.

Includes:
"sim_dfns.h"
"sim_types.h"

2.6.1.2.1 find_cubic_func

This routine generates a cubic function, given *x_0*, *y_0*, the X,Y coordinates of the break point (*x_b*, *y_b*), the X, Y coordinates of an end point (*x_f*, *y_f*), a factor related to the curvature of the function (*f*), and a pointer to an array which contains nine values (*func_arg[]*).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
x_0	REAL	sim_types.h
y_0	REAL	sim_types.h
x_b	REAL	sim_types.h
y_b	REAL	sim_types.h
x_f	REAL	sim_types.h
y_f	REAL	sim_types.h
f	REAL	sim_types.h
func_args[]	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
a_max	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	The function could not be generated
TRUE	int	The function was generated.

Table 2.6-2: find_cubic_func Information.

2.6.1.2.2 cubic_func

This routine finds a solution to the cubic function, given an x value and an array containing nine values used to generate the curve.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h
func_args[]	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
(val * (val * (val * func_args[5] + func_args[6]) + func_args[7]) + func_args[8])	REAL	the solution of the function at val (val > func_args[0])
(val * func_args[3] + func_args[4])	REAL	the solution of the function at val (val >= func_args[1])
(func_args[2] - (temp* (temp * (temp * func_args[5] + func_args[6]) +func_args[7]) + func_args[8])	REAL	the solution of the function at val

Table 2.6-3: cubic_func Information.

2.6.1.3 inv_sin_cos.c

(/simnet/release/src/libsrc/libmath /inv_sin_cos.c)

This file contains routines which return an angle in degrees or radians when given the sin and cosine of the angle.

Includes:

"sim_types.h"
"sim_dfns.h"

Defines:

HALF_PI
S_C_45
DEG_COEFF_3
DEG_COEFF_1
RAD_COEFF_3
RAD_COEFF_1

2.6.1.3.1 inv_sin_cos_deg

Given the sine and cosine of an angle, this routine returns the value of the angle in degrees.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
s	REAL	sim_types.h
c	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
(DEG_COEFF_3*s*s + DEG_COEFF_1)*s	REAL	angles between -45° and 45°
(DEG_COEFF_3*c*c+ DEG_COEFF_1)*c-90	REAL	angles between -135° and -45°
(-DEG_COEFF_3*c*c- DEG_COEFF_1)*c+90	REAL	angles between 45° and 135°
(-DEG_COEFF_3*s*s- DEG_COEFF_1)*s+180	REAL	angles between 135° and 180°
(-DEG_COEFF_3*s*s- DEG_COEFF_1)*s-180	REAL	angles between -180° and -135°

Table 2.6-4: inv_sin_cos_deg Information.

2.6.1.3.2 inv_sin_cos_rad

Given the sine and cosine of an angle, this routine returns the value of the angle in radians.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
s	REAL	sim_types.h
c	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
(RAD_COEFF_3*s*s + RAD_COEFF_1)*s	REAL	angles between -PI/4 and PI/4
(RAD_COEFF_3*c*c+ RAD_COEFF_1)*c-HALF_PI	REAL	angles between - 3*PI/4 and - PI/4
(-RAD_COEFF_3*c*c- RAD_COEFF_1)*c+HALF_PI	REAL	angles between PI/4 and 3*PI/4
(-RAD_COEFF_3*s*s- RAD_COEFF_1)*s+PI	REAL	angles between 3*PI/4 and PI
(-RAD_COEFF_3*s*s- RAD_COEFF_1)*s-PI	REAL	angles between -PI and - 3*PI/4

Table 2.6-5: inv_sin_cos_deg Information.

2.6.1.4 least_sq_fit.c
(/simnet/release/src/libsrc/libmath/least_sq_fit)

This file is an implementation of the least squares method for fitting a general polynomial:

$$y = a_0 + a_1 * x + a_2 * x^2 + \ldots + a_r * x^r$$

to a set of data points $<x_0,y_0>, <x_1,y_1>, \ldots, <x_n,y_n>$.

Any subset of the coefficients a_i is allowed to be fixed, and therefore there are at most r+1 parameters to solve for. Reference any engineering mathematics text for further information about this algorithm.

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"least_sq_fit.h"

Declarations:
The following are various matrices and arrays for intermediate values:
**x_powers
*x_powers_data
*x_powers_sums
**y_powers
*y_powers_data
*y_powers_sums
**sim_lin_eq
*sim_lin_eq_data
*sim_lin_eq_sums
*sim_lin_eq_sol

This is the rank, or dimension, of the matrix describing the simultaneous equations which must be solved:
rank

The following procedures are declared:
allocate_x_powers()
allocate_y_powers()
allocate_sim_lin_eq()
generate_x_powers()
generate_y_powers()
generate_sim_lin_eq()
solve_sim_lin_eq()
generate_output_coeff_vals()

2.6.1.4.1 least_squares_fit

This is the general algorithm. Given the general polynomial:

$$y = a_0 + a_1 * x + a_2 * x^2 + \ldots + a_r * x^r = \sum_{j=0}^{r} a_j * x^j$$

and the data points $<x_0,y_0>$, $<x_1,y_1>$, ... , $<x_n,y_n>$, each deviation, or residual, is of the form:

$$v_i = \sum_{j=0}^{r} a_j * x^j - y_i$$

with i=0,1,2...,n. The sum of the squares of the deviations is:

$$S = \sum_{i=0}^{n} v_i^2$$

Since S is a function of the a_k which are not fixed, S may be minimized by setting the partial derivatives of S with respect to each a_k equal to 0. The following results:

$$\frac{dS}{da_k} = 2 * \sum_{i=0}^{n} v_i * \frac{dv_i}{da_k} = 0$$

with k=0,1,...,r. Since the partial derivatives are:

$$\frac{dv_i}{da_k} = x_i^k$$

and dividing through by 2:

$$\sum_{i=0}^{n} v_i * x_i^k = 0$$

Substituting for v_i, the following is found:

$$\sum_{j=0}^{r} a_j * \sum_{i=0}^{n} x_i^{j+k} = \sum_{i=0}^{n} x_i^k y_i$$

with k=0,1,...,r. This can be written as:

$$\sum_{j=0}^{r} a_j * \sum_{i=0}^{n} x_i^{j+k} = \sum_{i=0}^{n} x_i y_i^k$$

with k=0,1,...,r. This represents an equation for each value of k which corresponds to a polynomial a_k which is not fixed. The result is simultaneous linear equations with the number of equations equal to the number of unknowns, which is equal to the number of non-fixed polynomial coefficients. The procedure **least_squares_fit**() must generate all the sums

$$\sum_{i=0}^{n} x_i^p$$

with p=0,1,...,2*r, and it must generate the sums

$$\sum_{i=0}^{n} x_i^{p*} y_i$$

with p=0,1,...,2*r. The routine then sets up the simultaneous linear equations in the variables a_k, and solves these equations to determine the coefficients of the polynomial.

The arguments to **least_squares_fit()** are as follows: *num_data_points* is equal to n+1. *x_vals* is an array of length *num_data_points* which stores x values. *y_vals* is an array of length *num_data_points* which stores y values. *poly_degree* is equal to r. *input_coeff_vals* is is an array of length *poly_degree* + 1 which stores coefficients of the polynomial. These coefficients are either UNKNOWN_COEFF or are fixed REAL numbers. *output_coeff_vals* is an array of length *poly_degree* + 1 which will hold the solution coefficients determined by this routine.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
num_data_points	int	Standard
poly_degree	int	Standard
x_vals	pointer to REAL	sim_types.h
y_vals	pointer to REAL	sim_types.h
input_coeff_vals	pointer to REAL	sim_types.h
output_coeff_vals	pointer to REAL	sim_types.h
Calls		
Function	**Where Described**	
allocate_x_powers	Section 2.6.1.4.2	
allocate_y_powers	Section 2.6.1.4.3	
allocate_sim_lin_eq	Section 2.6.1.4.4	
generate_x_powers	Section 2.6.1.4.5	
generate_y_powers	Section 2.6.1.4.6	
generate_sim_lin_eq	Section 2.6.1.4.7	
generate_output_coeff_vals	Section 2.6.1.4.9	

Table 2.6-6: least_squares_fit Information.

2.6.1.4.2 allocate_x_powers

This routine allocates memory for all of the sums:

$$\sum_{i=0}^{n} x_i^p$$

with p=0, 1, ..., 2*r. *num_data_points* is the number of data points, and *degree_poly* is the degree of the polynomial.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
num_data_points	int	Standard
poly_degree	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard

Table 2.6-7: allocate_x_powers Information.

2.6.1.4.3 allocate_y_powers

This routine allocates memory for all of the sums:

$$\sum_{i=0}^{n} x_i^{p*} y_i$$

with p=0, 1, ..., 2*r. *num_data_points* is the number of data points, and *degree_poly* is the degree of the polynomial.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
num_data_points	int	Standard
poly_degree	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard

Table 2.6-8: allocate_y_powers Information.

2.6.1.4.4 allocate_sim_lin_eq

This procedure allocates memory for simultaneous linear equations where the unknowns represent the unknown coefficients in the general polynomial. The rank of the matrix representing the simultaneous linear equations is equal to the number of unknown coefficients in the polynomial. *input_coeff_vals* represents the value of the input coefficients, and *degree_poly* is the degree of the polynomial.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
input_coeff_vals	pointer to REAL	sim_types.h
poly_degree	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
j	int	Standard

Table 2.6-9: allocate_sim_lin_eq Information.

2.6.1.4.5 generate_x_powers

This procedure generates all the sums:

$$\sum_{i=0}^{n} x_i^p$$

with p=0,1,...,2*r and where x_i^p is stored in position *x_powers[i][p]*, and the above sum is stored in *x_powers_sums[p]*. *num_data_points.* represents the number of data points; *degree_poly,* the degree of the polynomial. The input x values are represented by *x_vals.*

Parameters		
Parameter	**Type**	**Where Typedef Declared**
num_data_points	int	Standard
poly_degree	int	Standard
x_vals	pointer to REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
x_val	REAL	sim_types.h
x_power	REAL	sim_types.h
x_powers_sum	REAL	sim_types.h
i	int	Standard
j	int	Standard

Table 2.6-10: generate_x_powers Information.

2.6.1.4.6 generate_y_powers

This procedure generates all the sums:

$$\sum_{i=0}^{n} x_i^{p} * y_i$$

with p=0,1,...,2*r and where $x_i^p * y_i$ is stored in position *y_powers[i][p]*, and the above sum is stored in *y_powers_sums[p]*. The number of data points is represented by *num_data_points.* The degree of the polynomial is represented by *degree_poly.* The input y values are represented by *y_vals.*

Parameters		
Parameter	**Type**	**Where Typedef Declared**
num_data_points	int	Standard
poly_degree	int	Standard
y_vals	pointer to REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
y_powers_sum	REAL	sim_types.h
i	int	Standard
j	int	Standard

Table 2.6-11: generate_y_powers Information.

2.6.1.4.7 generate_sim_lin_eq

This procedure generates simultaneous linear equations. The equation:

$$\sum_{j=0}^{r} a_j * \sum_{i=0}^{n} x_i^{j+k} = \sum_{i=0}^{n} x_i^k y_i$$

with k=0,1,...,r, represents an equation for each value of k which corresponds to a polynomial coefficient ak which is not fixed. For each equation, the unknown aj's end up on the left hand side with coefficients taken from *x_powers_sums*, and the fixed aj's multiplied by their coefficients are subtracted from the *y_powers_sums* on the right hand side.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
num_data_points	int	Standard
poly_degree	int	Standard
vals	pointer to REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
j	int	Standard
k	int	Standard
eq_num	int	Standard
coeff_num	int	Standard

Table 2.6-12: generate_sim_lin_eq Information.

2.6.1.4.8 solve_sim_lin_eq

This procedure solves the simultaneous linear equations using the eleimination method, writing the solutions into the array *sim_lin_eq_sol.*

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
j	int	Standard
k	int	Standard
dividend	REAL	sim_types.h
factor	REAL	sim_types.h

Table 2.6-13: solve_sim_lin_eq Information.

2.6.1.4.9 generate_output_coeff_vals

This procedure places the solutions of the simultaneous linear equations into their proper positions in the general polynomial, skipping the polynomial coefficients that were initially fixed. The degree of the polynomial is represented by *degree_poly*. The input and output coefficients are represented by *input_coeff_vals* and *output_coeff_vals*, respectively.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
poly_degree	int	Standard
input_coeff_vals	pointer to REAL	sim_types.h
output_coeff_vals	pointer to REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
index	int	Standard

Table 2.6-14: generate_output_coeff_vals Information.

2.6.1.5 libmath.h

(/simnet/release/src/libsrc/libmath /libmath.h)

This file declares the following functions as external so that they can be used outside of libmath.

least_squares_fit()
bivariant_normal_distribution()
scaled_rand()
find_cubic_func()
cubic_func()
inv_sin_cos_rad()
inv_sin_cos_deg()
real_limit()
int_limit()

2.6.1.6 limit.c
(/simnet/release/src/libsrc/libmath /limit.c)

This file contains routines which limits the value that a parameter can have.

2.6.1.6.1 real_limit

This routine limits the value of the input parameter. *limit* is the limit that you wish to impose, and *input* is a pointer to the input parameter to be cropped.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
input	pointer to REAL	sim_types.h
limit	REAL	sim_types.h

Table 2.6-15: real_limit Information.

2.6.1.6.2 int_limit

This routine limits the value of the input parameter. *limit* is the limit that you wish to impose, and *input* is a pointer to the input parameter to be cropped.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
input	pointer to int	Standard
limit	int	Standard

Table 2.6-16: int_limit Information.

2.6.1.7 scaled_rand.c
(/simnet/release/src/libsrc/libmath /scaled_rand.h)

This file contains a function which generates a REAL random number between 0.0 and 1.0.

Includes:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
'sim_types.h"
"sim_macros.h"

Defines:
K32

2.6.1.7.1 scaled_rand

This routine generates a random REAL number between 0.0 and 1.0.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
random_number	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
random_number	REAL	a random number between 0.0 and 1.0

Table 2.6-17: scaled_rand Information.

2.6.2 libmatrix
(/simnet/release/src/libsrc/libmatrix [libmatrix])

The matrix library provides a set of routines for vector and matrix manipulation.

2.6.2.1 d2f_m_copy.c
(/simnet/release/src/libsrc/libmatrix/d2f_m_copy.c)

This file contains a routine, **d2f_mat_copy**, which copies a source matrix to a destination matrix.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.1.1 d2f_mat_copy

This routine copies a source matrix to a destination matrix. *src* is the source matrix, and *dst* is the destination matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
src [3] [3]	double	Standard
dst [3] [3]	float	Standard

Table 2.6-18: d2f_mat_copy Information.

2.6.2.2 d2f_v_copy.c

This file contains a routine, **d2f_v_copy**, which copies a source vector to a destination vector.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.2.1 d2f_vec_copy

This routine copies a source vector to a destination vector.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
src [3]	double	Standard
dst [3]	float	Standard

Table 2.6-19: d2f_v_copy Information.

2.6.2.3 elr_copy.c

This file contains a routine, **elr_copy**, which copies E_PARAM *from* to E_PARAM *to*.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.3.1 elr_copy

This routine copiers E_PAEAM *from* to E_PARAM *to*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
from	E_PARAM	sim_types.h
to	E_PARAM	sim_types.h

Table 2.6-20: elr_copy Information.

2.6.2.4 elr_elr_cat.c

This file contains one procedure, **elr_elr_cat**, which concatonates two Euler parameters.

The following files are included:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.4.1 elr_elr_cat

This routine concatonates two Euler parameters.

A_e_B0 are the Euler parameters for rotation from A to B0. *B0_e_B* are the Euler parameters for rotation from B0 to B. *A_e_B* are the Euler parameters for rotation from A to B.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
A_e_B0	E_PARAM	sim_types.h
B0_e_B	E_PARAM	sim_types.h
A_e_B	E_PARAM	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
mag	real	Standard

Table 2.6-21: elr_elr_cat Information.

2.6.2.5 elr_form.c

This file contains one procedure, **elr_form**, which creates E_PARAM given the axis of rotation and an angle.

This file includes:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.5.1 elr_form

This routine creates an E_PARAM given the axis of rotation and an angle.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
axis	VECTOR	sim_types.h
angle	REAL	sim_types.h
result	E_PARAM	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
sin_half_ang	REAL	sim_types.h

Table 2.6-22: elr_form Information.

2.6.2.6 elr_ident.c

This file contains one procedure, **elr_ident**, which initializes an E_PARAM to an identity transformation.

This file includes:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.6.1 elr_ident

This routine initializes an E_PARAM to an identity transformation.

Parameters		
Parameter	Type	Where Typedef Declared
e	E_PARAM	sim_types.h

Table 2.6-23: elr_ident Information.

2.6.2.7 elr_to_mat.c

This file contains one procedure, **elr_to_mat**, which converts from an E_PARAM to a T_MATRIX.

This file includes:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.7.1 elr_to_mat

This routine converts from an E_PARAM to a T_MATRIX. *A_e_B* is the quaternion for rotation of B to A. *A_c_B* is the direction cosine matrix from frame A to frame B. *temp* is a temporary variable.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
A_e_B	E_PARAM	sim_types.h
A_c_B	T_MATRIX	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp	REAL	sim_types.h
e0e0	REAL	sim_types.h
e0e1	REAL	sim_types.h
e0e2	REAL	sim_types.h
e0e3	REAL	sim_types.h
e1e1	REAL	sim_types.h
e1e2	REAL	sim_types.h
e1e3	REAL	sim_types.h
e2e2	REAL	sim_types.h
e2e3	REAL	sim_types.h
e3e3	REAL	sim_types.h

Table 2.6-24: elr_to_mat Information.

2.6.2.8 elr_transp.c

This file contains one procedure, **elr_transpose**, which transposes an E_PARAM into the *result* E_PARAM.

This file includes:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.8.1 elr_transpose

This routine transposes an E_PARAM into the result E_PARAM.

Parameters		
Parameter	Type	Where Typedef Declared
e	E_PARAM	sim_types.h
result	E_PARAM	sim_types.h

Table 2.6-25: elr_transpose Information.

2.6.2.9 f2d_m_copy

This file contains one procedure, **f2d_mat_copy**, which copies a source matrix to a destination matrix.

This file includes:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.9.1 f2d_mat_copy

This routine copies a source matrix to a destination matrix.

Parameters		
Parameter	Type	Where Typedef Declared
src [3] [3]	float	Standard
dst [3] [3]	double	Standard

Table 2.6-26: f2d_mat_copy Information.

2.6.2.10 f2d_v_copy.c

This file contains one procedure, **f2d_vec_copy**, which copies a source vector to a destination vector.

This file includes:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h

2.6.2.10.1 f2d_vec_copy

This routine copies a source vector to a destination vector.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
src [3]	float	Standard
dst [3]	double	Standard

Table 2.6-27: f2d_vec_copy Information.

2.6.2.11 fm_check.c

This file contains one procedure, **fmat_check**, which checks that all vectors in a matrix of floats are normalized.

This file includes:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.11.1 fmat_check

This routine checks to ensure that all vectors in a matrix of floats are normalized.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m [3] [3]	float	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
tmp [3] [3]	float	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	all vectors normalized
FALSE	int	not all vectors normalized
Calls		
Function	**Where Described**	
fmat_copy	Section 2.6.2.12 fm_copy.c	
fvec_check	Section 2.6.2.20 fv_check.c	
fmat_transpose	Section 2.6.2.19 fmat_transp.c	

Table 2.6-28: fmat_check Information.

2.6.2.12 fm_copy.c

This file contains one procedure, **fmat_copy**, which copies a float matrix *from* to a float matrix *to*.

This file includes:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.12.1 fmat_copy

This procedure copies a float matrix *from* to a float matrix *to*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
from [3] [3]	float	Standard
to [3] [3]	float	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
to_temp	register float	Standard
from_temp	register float	Standard
Calls		
Function	**Where Described**	
fmat_check	Section 2.6.2.11 .1	
fmat_transpose	Section 2.6.2.19 .1	

Table 2.6-29: fmat_copy Information.

2.6.2.13 fm_id_init.c

This file contains one procedure, **fmat_ident_init**, which initializes a matrix to be the identity matrix.

This file includes:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.13.1 fmat_ident_init

This procedure initializes a matrix to be the identity matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m [3] [3]	float	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
m_ptr	register float	Standard

Table 2.6-30: fmat_ident_init Information.

2.6.2.14 fm_m_mul.c

This file contains one procedure, **fmat_mat_mul**, which multiplies two float matrices together and stores the result.

This file includes:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.14.1 fmat_mat_mul

This procedure multiplies two float matrices together and stores the result.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m1 [3] [3]	float	Standard
m2 [3] [3]	float	Standard
result [3] [3]	float	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
m1_00	static float	Standard
m1_01	static float	Standard
m1_02	static float	Standard
m1_10	static float	Standard
m1_11	static float	Standard
m1_12	static float	Standard
m1_20	static float	Standard
m1_21	static float	Standard
m1_22	static float	Standard
m2_00	static float	Standard
m2_01	static float	Standard
m2_02	static float	Standard
m2_10	static float	Standard
m2_11	static float	Standard
m2_12	static float	Standard
m2_20	static float	Standard
m2_21	static float	Standard
m2_22	static float	Standard
mat_ptr	register float	Standard
res_ptr	register float	Standard

Table 2.6-31: fmat_mat_mul Information.

2.6.2.15 fm_r_init.c

This file contains one procedure, **fmat_rot_init**, which initializes a matrix to be a rotation matrix.

This file includes:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.15.1 fmat_rot_init

This procedure initializes a matrix to be a rotation matrix. Rotation is counter-clockwise when viewed along a positive axis.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m [3] [3]	float	Standard
theta	float	Standard
rot_axis	int	Standard

Table 2.6-32: fmat_rot_init Information.

2.6.2.16 fmat_dump.c

This file contains one procedure, **fmat_dump**, which dumps a matrix to the standard output.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.16.1 fmat_dump

This procedure prints a matrix and a descriptive message to the standard output.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
str	char	Standard
mat [3] [3]	float	Standard

Table 2.6-33: fmat_dump Information.

2.6.2.17 fmat_r_init2.c

This file contains one procedure, **fmat_rot_init2**, which initializes a matrix to be a rotation matrix.

This file includes:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.17.1 fmat_rot_init2

This procedure initializes a matrix to be a rotation matrix, given the sine of an angle instead of the angle itself. Rotation is counter-clockwise when viewed along a positive axis.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m [3] [3]	float	Standard
sin_theta	REAL	sim_types.h
rot_axis	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
cos_theta	float	Standard
one_minus_sqr_sin	float	Standard
m_ptr	register float	Standard
Calls		
Function	**Where Described**	
square	sim_macros.h (macro definition)	
abs	sim_macros.h (macro definition)	

Table 2.6-34: fmat_rot_init2 Information.

2.6.2.18 fmat_sub.c

This file contains one procedure, **fmat_sub**, which subtracts two matrices.

This file includes:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.18.1 fmat_sub

This procedure subtracts matrix *m2* from *m1*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m1 [3] [3]	float	Standard
m2 [3] [3]	float	Standard
result [3] [3]	float	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
m1_P	register float	Standard
m2_P	register float	Standard
result_P	register float	Standard

Table 2.6-35: fmat_sub Information.

2.6.2.19 fmat_transp.c

This file contains one procedure, **fmat_transpose**, which transposes a matrix into the *result* matrix.

The following files are included:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.19.1 fmat_transpose

This routine transposes a matrix into the *result* matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m [3] [3]	float	Standard
result [3] [3]	float	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp [3] [3]	float	Standard
temp_ptr	register float	Standard
Calls		
Function	**Where Described**	
fmat_copy	Section 2.6.2.12.1	
Called by		
Function	**Where Described**	
fmat_check	Section 2.6.2.11 fm_check.c	

Table 2.6-36: fmat_transpose Information.

2.6.2.20 fv_check.c

This file contains one procedure, **fvec_check**, which checks to see that a vector is normalized.

The following files are included:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.20.1 fvec_check

This routine checks to see that a vector is normalized.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v [3]	float	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	is normalized
FALSE	int	is not normalized
Calls		
Function	**Where Described**	
eq	sim_macros.h (macro definition)	
square	sim_macros.h (macro definition)	

Table 2.6-37: fvec_check Information.

2.6.2.21 fv_d_prod.c

This file contains one procedure, **fvec_dot_prod**, which computes a vector dot product and returns the result.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.21.1 fvec_dot_prod

This routine calculates a vector dot product and returns the result.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v1 [3]	float	Standard
v2 [3]	float	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp	register float	Standard
Return Values		
Return Value	**Type**	**Meaning**
temp	float	vector dot product result

Table 2.6-38: fvec_dot_prod Information.

2.6.2.22 fv_m_mul.c

This file contains one procedure, **fvec_mat_mul**, which multiplies a vector by a matrix and stores the result.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.22.1 fvec_mat_mul

This routine multiplies a vector by a matrix and stores the result..

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v [3]	float	Standard
result [3]	float	Standard
m [3] [3]	float	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
v0	static float	Standard
v1	static float	Standard
v2	static float	Standard
res_ptr	register float	Standard
vec_ptr	register float	Standard

Table 2.6-39: fvec_mat_mul Information.

2.6.2.23 fv_scale.c

This file contains one procedure, **fvec_scale**, which scales a vector.

The following files are included:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.23.1 fvec_scale

This routine scales a vector.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v [3]	float	Standard
result [3]	float	Standard
scale_factor	float	Standard

Table 2.6-40: fvec_scale Information.

2.6.2.24 fv_x_prod.c

This file contains one procedure, **fvec_cross_prod**, which computes a vector cross product and stores the result.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.24.1 fvec_cross_prod

This routine calculates a vector cross product (*v1* x *v2*) and stores the result. The right hand rule applies; the cross product sweeps from *v1* to *v2*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v1 [3]	float	Standard
v2 [3]	float	Standard
result [3]	float	Standard

Table 2.6-41: fvec_cross_prod Information.

2.6.2.25 fvec_add.c

This file contains one procedure, **fvec_add**, which adds two vectors.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.25.1 fvec_add

This routine adds two vectors.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v1 [3]	float	Standard
v2 [3]	float	Standard
result [3]	float	Standard

Table 2.6-42: fvec_add Information.

2.6.2.26 fvec_copy.c

This file contains one procedure, **fvec_copy**, which copies VECTOR *from* to VECTOR *to*.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.26.1 fvec_copy

This routine copies vector *from* to vector *to*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
from [3]	float	Standard
to [3]	float	Standard

Table 2.6-43: fvec_copy Information.

2.6.2.27 fvec_dump

This file contains one procedure, **fvec_dump**, which dumps a vector and accompanying message to the standard output.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.27.1 fvec_dump

This routine dumps a vector and accompanying message to the standard output.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
str	char	Standard
v [3] [3]	float	Standard

Table 2.6-44: fvec_dump Information.

2.6.2.28 fvec_norm.c

This file contains one procedure, **fvec_normalize**, which normalizes a vector.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.28.1 fvec_normalize

This routine normalizes a vector.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v [3]	float	Standard
result [3]	float	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp	register double	Standard
res_ptr	register float	Standard
v_ptr	register float	Standard
Calls		
Function	**Where Described**	
eq	sim_macros.h (macro definition)	

Table 2.6-45: fvec_normalize Information.

2.6.2.29 fvec_sub.c

This file contains one procedure, **fvec_sub**, which subtracts two vectors.

The following files are included:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.29.1 fvec_sub

This routine subtracts two vectors.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v1 [3]	float	Standard
v2 [3]	float	Standard
result [3]	float	Standard

Table 2.6-46: fvec_sub Information.

2.6.2.30 m_fix_m.c

This file contains routine, **mat_fix_matrix**, which restores a matrix to a state of orthonormality. *Axis* is the most important column of the matrix.

The following files are included:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.30.1 mat_fix_matrix

This routine restores a matrix to a state of orthonormality. *Axis* is the most important column of the matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mat	T_MATRIX	sim_types.h
axis	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
v_dot_w	REAL	sim_types.h
u	VECTOR	sim_types.h
v	VECTOR	sim_types.h
w	VECTOR	sim_types.h
temp	VECTOR	sim_types.h
r	int	Standard
Calls		
Function	**Where Described**	
vec_normalize	Section 2.6.2.65 .1	
vec_dot_prod	Section 2.6.2.56 .1	
vec_scale	Section 2.6.2.66 .1	
vec_add	Section 2.6.2.59 .1	
vec_cross_prod	Section 2.6.2.68 .1	

Table 2.6-47: mat_fix_matrix Information.

2.6.2.31 m_id_init.c

This file contains one procedure, **mat_ident_init**, which initializes a matrix to be the identity matrix.

The following files are included:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.31.1 mat_ident_init

This routine initializes a matrix to be the identity matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
m_ptr	REAL	sim_types.h

Table 2.6-48: mat_ident_init Information.

2.6.2.32 m_m_mul.c

This file contains one procedure, **mat_mat_mul**, which multiplies two matrices and stores the result.

The following files are included:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.32.1 mat_mat_mul

This routine multiplies two matrices and stores the result.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m1	REAL	sim_types.h
m2	T_MATRIX	sim_types.h
result	T_MATRIX	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
m1_x0	register REAL	sim_types.h
m1_x1	register REAL	sim_types.h
m1_x2	register REAL	sim_types.h
res_mat	T_MATRIX	sim_types.h
res_ptr	register REAL	sim_types.h
to	register long	Standard
from	register long	Standard

Table 2.6-49: mat_mat_mul Information.

2.6.2.33 m_r_int2.c

This file contains one procedure, **mat_rot_init2**, which initializes a matrix to be a rotation matrix.

The following files are included:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.33.1 mat_rot_init2

This routine initializes a matrix to be a rotation matrix. Rotation is counter-clockwise when viewed along a positive axis. The sine of the angle is given, instead of the angle itself.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	sim_types.h
sin_theta	REAL	sim_types.h
rot_axis	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
cos_theta	REAL	sim_types.h
m_ptr	register REAL	sim_types.h
Calls		
Function	**Where Described**	
abs	sim_macros.h *(macro definition)*	
square	sim_macros.h *(macro definition)*	

Table 2.6-50: mat_rot_init2 Information.

2.6.2.34 m_trig_init.c

This file contains procedure **mat_trig_init**, which initializes a matrix to be a rotation matrix, given the sine and cosine of an angle.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.34.1 mat_trig_init

This routine initializes a matrix to be a rotation matrix, given the sine and cosine of the angle, instead of the angle itself. Rotation is counter-clockwise when viewed along a positive axis.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	sim_types.h
sin_theta	REAL	sim_types.h
cos_theta	REAL	sim_types.h
rot_axis	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
m_ptr	register REAL	sim_types.h

Table 2.6-51: mat_trig_init Information.

2.6.2.35 m_v_mul.c

This file contains procedure **mat_vec_mul**, which multiplies a vector by a matrix and stores the result.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.35.1 mat_vec_mul

This routine multiplies a vector by a matrix and stores the result.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	sim_types.h
v	VECTOR	sim_types.h
result	VECTOR	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
v0	REAL	sim_types.h
v1	REAL	sim_types.h
v2	REAL	sim_types.h
res_ptr	register REAL	sim_types.h
vec_ptr	register REAL	sim_types.h

Table 2.6-52: mat_vec_mul Information.

2.6.2.36 mat_add.c

This file contains procedure **mat_add**, which adds two matrices.

The following files are included:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.36.1 mat_add

This routine adds two matrices and stores the result.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m1	T_MATRIX	sim_types.h
m2	T_MATRIX	sim_types.h
result	T_MATRIX	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
m1_P	register REAL	sim_types.h
m2_P	register REAL	sim_types.h
result_P	register REAL	sim_types.h

Table 2.6-53: mat_add Information.

2.6.2.37 mat_adj.c

This file contains one procedure, **mat_adjugate**, which calculates the adjugate matrix.

The following files are included:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.37.1 mat_adjugate

This routine calculates the adjugate matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	sim_types.h
result	T_MATRIX	sim_types.h

Table 2.6-54: mat_adjugate Information.

2.6.2.38 mat_check.c

This file contains one procedure, **mat_check**, which checks that all vectors in a matrix are normalized.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.38.1 mat_check

This routine checks that all vectors in a matrix are normalized.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
tmp	T_MATRIX	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	all normalized
FALSE	int	not all normalized
Calls		
Function	**Where Described**	
mat_copy	Section 2.6.2.39 .1	
vec_check	Section 2.6.2.52 .1	

Table 2.6-55: mat_check Information.

2.6.2.39 mat_copy.c

This file contains one procedure, **mat_copy**, which copies T_MATRIX *from* to T_MATRIX *to*.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.39.1 mat_copy

This routine copies T_MATRIX *from* to T_MATRIX *to*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
src	T_MATRIX	sim_types.h
dest	T_MATRIX	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
to	long	Standard
from	long	Standard

Table 2.6-56: mat_copy Information.

2.6.2.40 mat_deter.c

This file contains one procedure, **mat_determinant**, which returns the determinant of a matrix.

The following files are included:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.40.1 mat_determinant

This routine returns the determinant of a matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
(m[0][0]*m[1][1]*m[2][2]) + (m[0][1]*m[1][2]*m[2][0]) + (m[0][2]*m[1][0]*m[2][1]) + (m[0][0]*m[1][2]*m[2][1]) + (m[0][1]*m[1][0]*m[2][2]) + (m[0][2]*m[1][1]*m[2][0])	REAL	determinant of m

Table 2.6-57: mat_determinant Information.

2.6.2.41 mat_dump.c

This file contains one procedure, **mat_dump**, which dumps a matrix to the standard output.

The following files are included:
"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.41.1 mat_dump

This routine dumps a matrix and accompanying message to the standard output.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mat	T_MATRIX	sim_types.h
str	char	Standard

Table 2.6-58: mat_dump Information.

2.6.2.42 mat_form.c

This file contains one procedure, **mat_form**, which forms a T_MATRIX from an axis of rotation and an angle.

The following files are included:
"sim_types.h"
"math.h"

2.6.2.42.1 mat_form

This procedure forms a T_MATRIX from an axis of rotation and an angle.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
axis	VECTOR	sim_types.h
angle	REAL	sim_types.h
C	T_MATRIX	sim_types.h
sin_ang	REAL	sim_types.h
cos_ang	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
one_m_cos	REAL	sim_types.h
11_sin	REAL	sim_types.h

12_sin	REAL	sim_types.h
13_sin	REAL	sim_types.h
112	REAL	sim_types.h
113	REAL	sim_types.h
123	REAL	sim_types.h

Table 2.6-59: mat_form Information.

2.6.2.43 mat_ident.c

This file contains one procedure, **mat_ident**, which initializes a matrix to the unity matrix.

The following files are included:
"stdio.h"
"ctype.h"
"sim_types.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.43.1 mat_ident

This procedure initializes a matrix to the unity matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp	register REAL	sim_types.h

Table 2.6-60: mat_ident Information.

2.6.2.44 mat_init.c

This file contains procedure **mat_init**, which initializes all elements of a matrix to zero.

The following files are included:
"stdio.h"
"ctype.h"
"sim_types:h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.44.1 mat_init

This procedure initializes all elements of a matrix to zero.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp	register REAL	sim_types.h

Table 2.6-61: mat_init Information.

2.6.2.45 mat_inv.c

This file contains one procedure, **mat_inverse**, which calculates the inverse of a matrix.

The following files are included:
"stdio.h"
"ctype.h"
"sim_types.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.45.1 mat_inverse

This procedure calculates the inverse of a matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	sim_types.h
result	T_MATRIX	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
tmp	T_MATRIX	sim_types.h
Calls		
Function	**Where Described**	
mat_adjugate	Section 2.6.2.38 .1	
mat_transpose	Section 2.6.2.52 .1	
mat_scale	Section 2.6.2.49 .1	

Table 2.6-62: mat_inverse Information.

2.6.2.46 mat_lev_init.c

This file contains one procedure, **mat_level_init**, which forms a T_MATRIX whose Y axis is given and whose X axis is parallel to the ground.

The following files are included:
"stdio.h"
"ctype.h"
"sim_types:h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.46.1 mat_level_init

This routine forms a T_MATRIX whose Y axis is given and whose X axis is parallel to the ground.

m is the matrix to be initialized. *v* is the normalized vector which is the Y axis of m. *mptr* is a pointer to the elements in *m*. *vptr* is a pointer to the elements in *v*. *scale* is a pointer to a factor used to normalize the X axis.

The X axis is found by taking the cross product of *v* and the unit vector perpendicular to the ground, (0 0 1). The result must be normalized so *scale* (the magnitude of the result) is found. If *scale* is 0, *v* is pointing straight up or down. In this case, the X axis is defined to lay along the X axis of the parent system.

The Y axis is given, so *v* is copied onto *m*.

The Z axis is the cross product of the X and Y axes. Since they are perpendicular, the result is automatically normalized. It turns out that the last component of this vector is the magnitude of the cross product of the Y axis and the unit vertical. This was found above and stored as a result of *scale* being set as a pointer to *m [2] [2]*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	sim_types.h
v	VECTOR	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
mptr	register REAL	sim_types.h
vptr	register REAL	sim_types.h
scale	register REAL	sim_types.h

Table 2.6-63: mat_level_init Information.

2.6.2.47 mat_r_init.c

This file contains one procedure, **mat_rot_init**, which initializes a matrix to be a rotation matrix.

The following files are included:
- "stdio.h"
- "ctype.h"
- "sim_types.h"
- "math.h"
- "sim_dfns.h"
- "sim_macros.h"
- "libmatrix.h"

2.6.2.47.1 mat_rot_init

This procedure initializes a matrix to be a rotation matrix. Rotation is counter-clockwise when viewed along a positive axis.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	sim_types.h
theta	REAL	sim_types.h
rot_axis	int	Standard

Table 2.6-64: mat_rot_init Information.

2.6.2.48 mat_scale.c

This file contains one procedure, **mat_scale**, which scales a matrix.

The following files are included:
"stdio.h"
"ctype.h"
"sim_types.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.48.1 mat_scale

This procedure scales a matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mat	T_MATRIX	sim_types.h
result	T_MATRIX	sim_types.h
scale_factor	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
mat_p	register REAL	sim_types.h
result_P	register REAL	sim_types.h

Table 2.6-65: mat_scale Information.

2.6.2.49 mat_sub.c

This file contains one procedure, **mat_sub,** which subtracts two matrices.

The following files are included:
"stdio.h"
"ctype.h"
"sim_types.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.49.1 mat_sub

This procedure subtracts two matrices.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m1	T_MATRIX	sim_types.h
m2	T_MATRIX	sim_types.h
result	T_MATRIX	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
m1_P	register REAL	sim_types.h
m2_P	register REAL	sim_types.h
result_P	register REAL	sim_types.h

Table 2.6-66: mat_sub Information.

2.6.2.50 mat_to_elr.c

This file contains one procedure, **mat_to_elr**, which converts from T_MATRIX to E_PARAM.

The following files are included:
"stdio.h"
"ctype.h"
"sim_types.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.50.1 mat_to_elr

This procedure converts from T_MATRIX to E_PARAM. The Euler parameter is made from the direction cosine matrix. REAL_SMALL is defined as 1.0e-35.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
C	T_MATRIX	sim_types.h
e	E_PARAM	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
denominator	REAL	sim_types.h

Table 2.6-67: mat_to_elr Information.

2.6.2.51 mat_transp.c

This file contains one procedure, **mat_transpose,** which transposes a matrix into the result matrix.

The following files are included:

"stdio.h"
"ctype.h"
"sim_types.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.51.1 mat_transpose

This procedure transposes a matrix into the result matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	sim_types.h
result	T_MATRIX	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp	T_MATRIX	sim_types.h
temp_ptr	register REAL	sim_types.h
Calls		
Function	**Where Described**	
mat_copy	Section 2.6.2.40 .1	

Table 2.6-68: mat_transpose Information.

2.6.2.52 new_m_m_mul.c

This file contains one procedure, **nmat_mat_mul,** which multiplies two matrices.

The following files are included:
"stdio.h"
"ctype.h"
"sim_types.h"
"math.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.52.1 nmat_mat_mul

This procedure multiplies two matrices and stores the result.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m1	register REAL	sim_types.h
m2	register T_MATRIX	sim_types.h
result	T_MATRIX	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
m1_x0	register REAL	sim_types.h
m1_x1	register REAL	sim_types.h
m1_x2	register REAL	sim_types.h
res_mat	T_MATRIX	sim_types.h
res_ptr	register REAL	sim_types.h
to	register long	Standard
from	register long	Standard

Table 2.6-69: nmat_mat_mul Information.

2.6.2.53 v_cos_prod.c

This file contains one procedure, **vec_cos_prod**, which computes the cosine of the angle between two vectors.

The following files are included:
"stdio.h"
"ctypes.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.53.1 vec_cos_prod

This procedure computes the cosine of the angle between two vectors.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v1	VECTOR	sim_types.h
v2	VECTOR	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
len1	register REAL	sim_types.h
len2	register REAL	sim_types.h
vec_dot_prod()	REAL	sim_types.h
result	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
result	REAL	cosine of the angle between v1 and v2
Calls		
Function	**Where Described**	
vec_dot_prod	Section 2.6.2.56 .1	

Table 2.6-70: vec_cos_prod Information.

2.6.2.54 v_dot_prod.c

This file contains one procedure, **vec_dot_prod**, which computes the vector dot product and returns the result.

The following files are included:

"stdio.h"
"ctypes.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.54.1 vec_dot_prod

This procedure computes the vector dot product and returns the result.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v1	VECTOR	sim_types.h
v2	VECTOR	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp	register REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
temp	REAL	dot product result

Table 2.6-71: vec_dot_prod Information.

2.6.2.55 v_e_transf.c

This file contains one procedure, **vec_elr_transform**, which transforms a vector by an Euler parameter.

The following files are included:
- "stdio.h"
- "ctypes.h"
- "math.h"
- "sim_types.h"
- "sim_dfns.h"
- "sim_macros.h"
- "libmatrix.h"

2.6.2.55.1 vec_elr_transform

This procedure transforms a vector by an Euler parameter. v_A is the source vector. A_to_B is the Euler parameter. v_B is the transformed vector.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v_A	VECTOR	sim_types.h
A_to_B	E_PARAM	sim_types.h
v_B	VECTOR	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp1	VECTOR	sim_types.h
temp2	VECTOR	sim_types.h
Calls		
Function	**Where Described**	
vec_cross_prod	Section 2.6.2.68 .1	
vec_scale	Section 2.6.2.66 .1	
vec_add	Section 2.6.2.59 .1	

Table 2.6-72: vec_elr_transform Information.

2.6.2.56 v_m_mul.c

This file contains one procedure, **vec_mat_mul**, which multiplies a vector by a matrix and stores the result.

The following files are included:

"stdio.h"
"ctypes.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.56.1 vec_mat_mul

This procedure multiplies a vector by a matrix and stores the result.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v	VECTOR	sim_types.h
m	T_MATRIX	sim_types.h
result	VECTOR	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
v0	static REAL	sim_types.h
v1	static REAL	sim_types.h
v2	static REAL	sim_types.h
res_ptr	register REAL	sim_types.h
vec_ptr	register REAL	sim_types.h

Table 2.6-73: vec_mat_mul Information.

2.6.2.57 vec_add.c

This file contains one procedure, **vec_add**, which adds two vectors.

The following files are included:

"stdio.h"
"ctypes.h"
"math.h"
"sim_types"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.57.1 vec_add

This procedure adds two vectors.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v1	VECTOR	sim_types.h
v2	VECTOR	sim_types.h
result	VECTOR	sim_types.h

Table 2.6-74: vec_add Information.

2.6.2.58 vec_check.c

This file contains one procedure, **vec_check**, which checks to see if a vector is normalized.

The following files are included:
"stdio.h"
"ctypes.h"
"math.h"
"sim_types"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.58.1 vec_check

This procedure checks to see if a vector is normalized.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v	VECTOR	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	is normalized
FALSE	int	*not normalized*
Calls		
Function	**Where Described**	
eq	sim_macros.h (macro definition)	
square	sim_macros.h (macro definition)	

Table 2.6-75: vec_check Information.

2.6.2.59 vec_copy.c

This file contains one procedure, **vec_copy**, which copies VECTOR *from* to VECTOR *to*.

The following files are included:
"stdio.h"
"ctypes.h"
"math.h"
"sim_types"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.59.1 vec_copy

This procedure copies VECTOR *from* to VECTOR *to*.

Parameters		
Parameter	Type	Where Typedef Declared
from	VECTOR	sim_types.h
to	VECTOR	sim_types.h

Table 2.6-76: vec_copy Information.

2.6.2.60 vec_dump.c

This file contains one procedure, **vec_dump**, which dumps a vector to the standard output.

The following files are included:
"stdio.h"
"ctypes.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.60.1 vec_dump

This procedure dumps a vector and accompanying message to the standard output.

Parameters		
Parameter	Type	Where Typedef Declared
v	VECTOR	sim_types.h
str	char	Standard

Table 2.6-77: vec_dump Information.

2.6.2.61 vec_init.c

This file contains one procedure, **vec_init**, which initializes a vector to zero.

The following files are included:
"stdio.h"
"ctypes.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.61.1 vec_init

This procedure initializes a vector to all zeroes.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v	VECTOR	sim_types.h

Table 2.6-78: vec_init Information.

2.6.2.62 vec_neg.c

This file contains one procedure, vec_neg, which computes -*v1*.

The following files are included:
"stdio.h"
"ctypes.h"
"math.h"
"sim_types"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.62.1 vec_neg

This procedure computes -*v1*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v1	VECTOR	sim_types.h
result	VECTOR	sim_types.h

Table 2.6-79: vec_neg Information.

2.6.2.63 vec_norm.c

This file contains one procedure, **vec_normalize**, which normalizes a vector.

The following files are included:
"stdio.h"
"ctypes.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.63.1 vec_normalize

This procedure normalizes a vector.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v	VECTOR	sim_types.h
result	VECTOR	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp	register double	Standard
res_ptr	register REAL	sim_types.h
v_ptr	register REAL	sim_types.h
Calls		
Function	**Where Described**	
eq	sim_macros.h (macro definition)	

Table 2.6-80: vec_normalize Information.

2.6.2.64 vec_scale.c

This file contains one procedure, vec_scale, which scales a vector.

The following files are included:
"stdio.h"
"ctypes.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.64.1 vec_scale

This procedure scales a vector.

Parameters		
Parameter	Type	Where Typedef Declared
v	VECTOR	sim_types.h
result	VECTOR	sim_types.h
scale_factor	REAL	sim_types.h

Table 2.6-81: vec_scale Information.

2.6.2.65 vec_sub.c

This file contains one procedure, vec_sub, which subtracts two vectors and stores the result.

The following files are included:
"stdio.h"
"ctypes.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.65.1 vec_sub

This procedure subtracts two vectors and stores the result.

Parameters		
Parameter	Type	Where Typedef Declared
v1	VECTOR	sim_types.h
v2	VECTOR	sim_types.h
result	VECTOR	sim_types.h

Table 2.6-82: vec_sub Information.

2.6.2.66 vec_x_prod.c

This file contains one procedure, *vec_cross_prod*, which computes a vector cross product and stores the result.

The following files are included:

"stdio.h"
"ctype.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"

2.6.2.66.1 vec_cross_prod

This routine computes a vector cross product and stores the result.

Parameters		
Parameter	Type	Where Typedef Declared
v1	VECTOR	sim_types.h
v2	VECTOR	sim_types.h
result	VECTOR	sim_types.h
Internal Variables		
Internal Variable	Type	Where Typedef Declared
temp	VECTOR	sim_types.h
Calls		
Function	Where Described	
vec_copy	Section 2.6.2.61 .1	

Table 2.6-83: vec_cross_prod Information.

2.6.2.67 libmatrix.h
(simnet/release/src/libsrc/libmatrix/libmatrix.h)

This file declares the routines found in libmatrix for use inside and outside of libmatrix.

2.6.3 libtimers
(./simnet/release/src/libsrc/libtimers [libtimers])

This CSU contains the routines which initialize the elapsed time clock at the startup of a simulation. It also provides functionality for an alarm clock tool. This allows for the timed sequencing of simulation events.

2.6.3.1 t_cur_tick.c
(./simnet/release/src/libsrc/libtimers/t_cur_tick.c)

This file contains one procedure, **timers_get_current_tick**, which returns the current elapsed time in ticks.

The following files are included:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.1.1 timers_get_current_tick

This procedure returns *libtimers_elapsed_ticks*, the elapsed time in ticks.

Return Values		
Return Value	**Type**	**Meaning**
libtimers_elapsed-ticks	int	elapsed time in ticks

Table 2.6-84: timers_get_current_tickInformation.

2.6.3.2 t_cur_time.c
(./simnet/release/src/libsrc/libtimers/t_cur_tick.c)

This file contains a routine which determines the elapsed time in seconds.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.2.1 timers_get_current_time

This procedure returns *libtimers_elapsed_time*, the elapsed time in seconds.

Return Values		
Return Value	**Type**	**Meaning**
libtimers_elapsed_time	REAL	elapsed time

Table 2.6-85: timers_get_current_time Information.

2.6.3.3 t_data.c
(./simnet/release/src/libsrc/libtimers/t_data.c)

This file contains a routine which returns information about the timers.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.3.1 timers_get_data

This procedure returns information about the timers.

Return Values		
Return Value	**Type**	**Meaning**
libtimers_data	REAL	information about the timers

Table 2.6-86: timers_get_data Information.

2.6.3.4 t_del_proc.c
(./simnet/release/src/libsrc/libtimers/t_del.c)

This file contains one routine which sets a timer to delay the implementation of a procedure for a specified amount of time.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.4.1 timers_delay_proc

This routine sets a timer to delay the implementation of a procedure for a specified amount of time. *ticks* is the time delay. *proc* is the proceedure of interest.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ticks	int	Standard
proc	PFI	sim_types.h
necessary	int	Standard
data	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
index	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
-1	int	unable to set timer
index	int	the timer set
Calls		
Function	**Where Described**	
timers_get_timer	Section 2.6.3.6.1	

Table 2.6-87: timers_delay_proc Information.

2.6.3.5 t_free.c
(./simnet/release/src/libsrc/libtimers/t_free.c)

This file contains a routine which frees a timer and resets its values so that it can be used again.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.5.1 timers_free_timer

This procedure contains one parameter, *index*, which is the timer number. It resets the *libtimers_timer_values* for a particular timer to a value set when they are not in use.

libtimers_timer_values[index].ticks_left
libtimers_timer_values[index].ticking_status
libtimers_timer_values[index].timeout_edge
libtimers_timer_values[index].in_use_status
libtimers_timer_values[index].stopped_status
libtimers_timer_values[index].proc
libtimers_timer_values[index].data

Parameters		
Parameter	Type	Where Typedef Declared
index	int	Standard

Table 2.6-88: timers_free_timer Information.

2.6.3.6 t_get_timer.c
(./simnet/release/src/libsrc/libtimers/t_get_timer.c)

This file contains a routine which sets the timer.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.6.1 timers_get_timer

This routine sets the timer, if given the number of ticks required to set the time.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ticks	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
index	int	Standard
in_use	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
-1	int	too many timers already set
index	int	the timer that is set

Table 2.6-89: timers_get_timer Information.

2.6.3.7 t_in_use.c
(./simnet/release/src/libsrc/libtimers/t_in_use.c)

This file contains a routine which determines if a timer is in use.

The following files are included:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.7.1 timers_get_in_use_status

This routine indicates whether or not a timer is currently in use. *index* is the timer of interest.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
libtimers_timer_values [index] .in_use_status	int	the timer's in use status
FREE	int	timer is free

Table 2.6-90: timers_get_in_use_status Information.

2.6.3.8 t_init.c
(./simnet/release/src/libsrc/libtimers/t_init.c.c)

This file contains a routine which sets the initial time at the onset of the simulaiton. It initializes all timers so that they are available for use.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"
"sys/types.h"
"sys/time.h"

2.6.3.8.1 timers_init

This routine sets the initial time at the start of a simulation and initializes all timers so that they are available for use. If a Masscomp machine is used, the start time is set by calling **ftime**(&libtimers_start_time). If this function returns -1, the starting time could not be set. If a Butterfly machine is used, the start time is set equal to *rtc*.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
timer_ptr	register pointer to *TIMER*	timers_dfn.h
index	register int	Standard

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
-1	int	start time could not be set

Calls	
Function	**Where Described**
ftime (if Masscomp)	Standard C function available on the Masscomp Machine

Table 2.6-91: timers_init Information.

2.6.3.9 t_loc.c
(./simnet/release/src/libsrc/libtimers/t_loc.c.c)

This file declares certain local variables and sets values to some of them.

2.6.3.10 t_milli.c
(./simnet/release/src/libsrc/libtimers/t_milli.c)

This file contains one routine which returns the elapsed time in milliseconds.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"
"net/network.h"

If a Masscomp machine is used, "sys/types.h" and "sys/timeb" are also included.

2.6.3.10.1 timers_elapsed_milliseconds

This procedure determines and returns the elapsed time in milliseconds.

If a Masscomp machine is used:

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
current_real_time	timeb	Masscomp types
diff_in_secs	time_t	Masscomp types
diff_in_millisecs	unsigned short	Standard
elapsed_millisecs	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
-1	int	can't get real time
elapsed_millisecs	int	elapsed time in milliseconds

Table 2.6-92: timers_elapsed_milliseconds Information for the Masscomp.

If a Butterfly machine is used:

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
current_real_millisec	long	Standard
Return Values		
Return Value	**Type**	**Meaning**
current_real_millisec - libtimers_start_millisec	int	elapsed time in milliseconds

Table 2.6-93: timers_elapsed_milliseconds Information for the Butterfly.

2.6.3.11 t_null_proc.c
(./simnet/release/src/libsrc/libtimers/t_null_proc.c)

This file contains a routine which is called by **timers_delay_proc()** to initialize the proceedure to be called after the timer has expired.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.11.1 timers_null_proc

This routine is called by **timers_delay_proc()** to initialize the procedure that will be called when the timer has expired.

Return Values		
Return Value	Type	Meaning
0	int	null

Table 2.6-94: timers_null_proc Information.

2.6.3.12 t_reset.c
(./simnet/release/src/libsrc/libtimers/t_reset.c)

This file contains a routine which resets the timeout edge so that the timer can be reused.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.12.1 timers_reset_timeout_edge

This function has one parameter, *index* , which represents the timer of interest. This procedure sets the *libtimers_timer_values[index] . timeout_edge* to *OFF*, so that the timer can be reused.

Parameters		
Parameter	Type	Where Typedef Declared
index	int	Standard

Table 2.6-95: timers_reset_timeout_edge Information.

2.6.3.13 t_restart.c

(./simnet/release/src/libsrc/libtimers/t_restart.c)

This file contains a routine which resets the stopped status of the timer to *FOREGROUND*.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.13.1 timers_restart_timer

This routine sets the stopped status of the timer to *FOREGROUND*. This causes the timer to resume elapsing time. This function has one parameter, *index* , which denotes the timer to be started.

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

Table 2.6-96: timers_restart_timer Information.

2.6.3.14 t_set_null.c

(./simnet/release/src/libsrc/libtimers/t_set_null.c)

This file contains a routine which sets an index to no timer.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.14.1 timers_set_null_timer

This routine sets an index to no timer and returns an indication of this.

Return Values		
Return Value	**Type**	**Meaning**
NULL_TIMER	int	index set to no timer

Table 2.6-97: timers_set_null_timer Information.

2.6.3.15 t_simul.c

(./simnet/release/src/libsrc/libtimers/t_simul.c)

This file contains a routine which keeps all of the timers up to date.

The following files are included:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"
"net/network.h"

2.6.3.15.1 timers_simul

This routine contains the functionality to coordinate all timers and keep them current.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
timer_ptr	register pointer to TIMER	timers_dfn.h
index	register int	Standard
Calls		
Function	**Where Described**	
timers_free_timer	Section 2.6.3.5.1	

Table 2.6-98: timers_simul Information.

2.6.3.16 t_start.c

(./simnet/release/src/libsrc/libtimers/t_start.c)

This file contains a routine which grabs the time from the operating system to set the start time for the timers.

The following files are included:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"
"net/network.h"

If a Masscomp machine is used, "sys/types.h" and "sys/timeb" are also included.

2.6.3.16.1 timers_init_starttime

This procedure initializes the start time for all timers by grabbing the current time from the operating system.

If a Masscomp machine is used, the function **ftime**(&libtimers_start_time) is called to set the start time for all timers. If **ftime** returns -1, the start time couldn't be set.

If a Butterfly machine is used, *libtimers_start_millisec* is set equal to *rtc*.

Calls	
Function	**Where Described**
ftime (if Masscomp)	Standard function available on the Masscomp.

Table 2.6-99: timers_init_starttime Information.

2.6.3.17 t_status.c
(./simnet/release/src/libsrc/libtimers/t_statuss.c)

This file contains a procedure which determines the status of all timers.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.17.1 timers_status

This procedure prints the status of all timers in use, and prints the number of timers counted. This is a debugging tool.

Internal Variables		
Internal Variable	Type	Where Typedef Declared
ret	int	Standard
i	int	Standard

Table 2.6-100: timers_status Information.

2.6.3.18 t_stop.c
(./simnet/release/src/libsrc/libtimers/t_stop.c)

This file contains a routine which allows you to stop a timer.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.18.1 timers_stop_timer

This procedure sets the value of *libtimers_timer_values[index] . stopped_status* to *STOPPED*, which stops that particular timer designated by *index*.

Parameters		
Parameter	Type	Where Typedef Declared
index	int	Standard

Table 2.6-101: timers_stop_timer Information.

2.6.3.19 t_stopped.c
(./simnet/release/src/libsrc/libtimers/t_stopped.c)

This file contains a procedure which returns the stopped status of a timer.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.19.1 timers_get_stopped_status

This procedure retuns the stopped status for the timer specified by *index*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
libtimers_timer_values [index] . stopped_status	int	stopped status of the timer
FOREGROUND	int	timer is running

Table 2.6-102: timers_get_stopped_status Information.

2.6.3.20 t_ticking.c

(./simnet/release/src/libsrc/libtimers/t_ticking.c)

This file contains a procedure which returns the ticking status of a given timer.

The following files are included:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.20.1 timers_get_ticking_status

This routine returns the ticking status of a timer specified by *index.*

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
libtimers_timer_values [index] . ticking_status	int	timer status
TIMED_OUT	int	timer timed out

Table 2.6-103: timers_get_ticking_status Information.

2.6.3.21 t_ticks.c
(./simnet/release/src/libsrc/libtimers/t_ticks.c)

This file contains one procedure, **timers_get_ticks_left()** which returns the number of ticks remaining.

The following files are included:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.21 timers_get_ticks_left.

This routine returns the number of ticks remaining on the timer designated by *index*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
libtimers_timer_values [index] . ticks_left	int	the number of ticks left
0	int	no more ticks left

Table 2.6-104: timers_get_ticks_left Information.

2.6.3.22 t_timeout.c
(./simnet/release/src/libsrc/libtimers/t_timeout.c)

This file contains a routine which returns the timeout edge for a given timer.

Includes:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers_dfn.h"
"timers.h"
"timers_loc.h"

2.6.3.22.1 timers_get_timeout_edge

This routine returns the timeout edge for a given timer. The routine returns *TRUE* on the tick that the timer went off and returns *FALSE* otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
index	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
libtimers_timer_values [index] .timeout_edge	int	TRUE if the timer went off on that tick FALSE if timer is ticking or off
0	int	timer not defined

Table 2.6-105: timers_get_timeout_edge Information.

2.6.3.23 timers_loc.h
(./simnet/release/src/libsrc/libtimers/timers_loc.h)

This file declares several external variables for use in routines within this library.

2.6.4 libutil

(/simnet/release/src/libsrc/libutil [libutil])

This library contains a collection of utilities which include clear screen and produce an audible prompt. This library also contains various copy procedures and formatted and/or timed printing routines.

2.6.4.1 beep.c

(/simnet/release/src/libsrc/libutil/beep.c)

This file contains one procedure, **beep**, which causes an audible prompt to be produced.

2.6.4.1.1 beep

This routine produces an audible prompt. *count* is an integer which is used as a counter.

Parameters		
Parameter	Type	Where Typedef Declared
count	int	Standard

Table 2.6-106: beep Information.

2.6.4.2 cp_2_TF1.c

(/simnet/release/src/libsrc/libutil/cp_2_TF1.c)

This file contains one routine, copy_to_TF1, which copies a T_MATRIX or a VECTOR to a TF1 structure.

Includes:
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"

2.6.4.2.1 copy_to_TF1

This routine copies a T_MATRIX or a VECTOR to a TF1 structure. *src_mtx* is a source matrix of type T_MATRIX. *src_vec* is a source vector of type VECTOR. *dst* is a pointer to the destination TF1.

Parameters		
Parameter	Type	Where Typedef Declared
src_mtx	T_MATRIX	sim_types.h
src_vec	VECTOR	sim_types.h
dst	pointer to TF1	sim_cig_if.h
Internal Variables		
Internal Variable	Type	Where Typedef Declared
i	int	Standard
j	int	Standard

Table 2.6-107: copy_to_TF1 Information.

2.6.4.3 cp_R4P3D.c
(/simnet/release/src/libsrc/libutil /cp_R4P3D.c)

This file contains a routine which copies a source R4P3D matrix to a destination R4P3D matrix.

The following are included:
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"

2.6.4.3.1 copy_R4P3D

This routine copies a source R4P3D matrix to a destination R4P3D matrix. *src* is the pointer to the source R4P3D matrix. *dst* is the pointer to the destination R4P3Dmatrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
src	pointer to R4P3D	dgi_stdg.h
dst	pointer to R4P3D	dgi_stdg.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard

Table 2.6-108: copy_R4P3D Information.

2.6.4.5 cp_TF1.c

(/simnet/release/src/libsrc/libutil/cp_TF1.c)

This file contains one routine, **copy_TF1**, which copies a source matrix and vector to destination matrix and vector.

The following are included:
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"

2.6.4.5.1 copy_TF1

This routine copies source matrix and vector to destination matrix and vector. *src* is a pointer to the source TF1 matrix, *dst* is a pointer to the destination TF1 matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
src	pointer to TF1	sim_cig_if.h
dst	pointer to TF1	sim_cig_if.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
j	int	Standard

Table 2.6-109: copy_TF1 Information.

2.6.4.6 cp_TF2.c

(/simnet/release/src/libsrc/libutil/cp_TF2.c)

The following are included:
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"

2.6.4.6.1 copy_TF2

This routine copies a source vector to a destination vector. *src* is a pointer to the source TF2 vector, and *dst* is a pointer to the destination TF2 vector.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
src	pointer to TF2	sim_cig_if.h
dst	pointer to TF2	sim_cig_if.h

Table 2.6-110: copy_TF2 Information.

2.6.4.7 cp_Xrot2TF2.c
(/simnet/release/src/libsrc/libutil/cp_Xrot2TF2.c)

This file contains a routine which copies a T_MATRIX matrix to a TF2 matrix.

Includes:
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"

2.6.4.7.1 copy_X_rot_to_TF2

The following routine copies a T_MATRIX matrix (*src*) to a TF2 matrix (dst).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
src	T_MATRIX	sim_types.h
dst	pointer to TF2	sim_cig_if.h

Table 2.6-111: copy_X_rot_to_TF2 Information.

2.6.4.8 cp_Yrot2TF2.c
(/simnet/release/src/libsrc/libutil/cp_Yrot2TF2.c)

This file contains a routine which copies a matrix of type T_MATRIX to a matrix of type TF2.

Includes:
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"

2.6.4.8.1 copy_Y_rot_to_TF2

The following routine copies a matrix of type T_MATRIX (*src*) to a matrix of type TF2 (dst).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
src	T_MATRIX	sim_types.h
dst	pointer to TF2	sim_cig_if.h

Table 2.6-112: copy_Y_rot_to_TF2 Information.

2.6.4.9 cp_Zrot2TF2.c
(/simnet/release/src/libsrc/libutil/cp_Zrot2TF2.c)

This file contains a routine which copies a matrix of type T_MATRIX to a TF2.

Includes:
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"

2.6.4.9.1 copy_Z_rot_to_TF2

The following routine copies a T_MATRIX (*src*) to a matrix of type TF2 (dst).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
src	T_MATRIX	sim_types.h
dst	pointer to TF2	sim_cig_if.h

Table 2.6-113: copy_Z_rot_to_TF2 Information.

2.6.4.10 database.c
(/simnet/release/src/libsrc/libutil/database.c)

This file contains routines which handle database names.

Includes:
"strings.h"
"ctype.h"

The following is declared:
database_in_use[14]

2.6.4.10.1 util_set_database_name

This routine translates a database name into a standard format string. *db* is a pointer to the database name.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
db	pointer to char	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
cp	pointer to char	Standard

Table 2.6-114: util_set_database_name Information.

2.6.4.10.2 util_get_database_name

This routine gets the name of the database in use.

Return Values		
Return Value	**Type**	**Meaning**
database_in_use	pointer to char	name of database in use

Table 2.6-115: util_get_database_name Information.

2.6.4.11 dead_zone.c
(/simnet/release/src/libsrc/libutil/dead_zone.c)

This file contains a routine which adds a "dead zone" to a control.

Includes:
"sim_types.h"
"sim_macros.h"
"sim_dfns.h"

2.6.4.11.1 add_dead_zone

This routine adds a zone where the controls don't respond. *control* is the control to be changed and *dead_zone* is the range to be designated as a "dead zone."

Parameters		
Parameter	**Type**	**Where Typedef Declared**
control	REAL	sim_types.h
dead_zone	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
control	REAL	sim_types.h
Calls		
Function	**Where Described**	
max	sim_macros.h (macro definition)	
min	sim_macros.h (macro definition)	

Table 2.6-116: add_dead_zone Information.

2.6.4.12 deg.c
(/simnet/release/src/libsrc/libutil/deg.c)

This file contains a routine which generates an angle given its sine and cosine.

Includes:
"sim_types.h"
"sim_macros"
"sim_dfns.h"

The following are defined:
S_C_45
COEFF_3
COEFF_1

2.6.4.12.1 sin_cos_to_deg

This routine returns the angle (in degrees) given the sine and cosine of that angle. *s* is the sine of the angle and *c* is the cosine of the angle.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
s	REAL	sim_types.h
c	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
((COEFF_3 * s * s + COEFF_1) * s)	REAL	the angle in degrees for angles between -45° and 45°
((COEFF_3 * c * c + COEFF_1) *c - 90.0)	REAL	the angle in degrees for angles between -135° and -45°
((-COEFF_3 * c * c - COEFF_1) *c + 90.0)	REAL	the angle in degrees for angles between 45° and 135°
((-COEFF_3 * s * s - COEFF_1) * s + 180.0)	REAL	the angle in degrees for angles between 135° and 180°
((-COEFF_3 * s * s -COEFF_1) * s -180.0)	REAL	the angle in degrees for angles between -180° and -135°

Table 2.6-117: sin_cos_to_deg Information.

2.6.4.13 dump_core.c
(/simnet/release/src/libsrc/libutil/dump_core.c)

This file contains a routine which purposely dumps core.

2.6.4.13.1 dump_core

This routine purposely dumps the core.

Internal Variables		
Internal Variable	Type	Where Typedef Declared
a	int	Standard

Table 2.6-118: dump_core Information.

2.6.4.14 error_printf.c
(/simnet/release/src/libsrc/libutil/error_printf.c)

This file contains a routine which prints an error report.

Includes:
"stdio.h"

2.6.4.14.1 error_printf

This routine prints a report. *function_name* is the function name where the error occurred. *ctl* is the printout control. *args[]* is the list of arguments. *text[150]* is the string to form the message in.

Parameters		
Parameter	Type	Where Typedef Declared
function_name	pointer to char	Standard
ctl	pointer to char	Standard
args[]	int	Standard
Internal Variables		
Internal Variable	Type	Where Typedef Declared
text[150]	char	Standard

Table 2.6-119: error_printf Information.

2.6.4.15 format.c
(/simnet/release/src/libsrc/libutil/format.c)

This file contains routines that when given a buffer, a format string, and a pointer to a variable list of arguments, will then fill the buffer with an ascii string that corresponds to the argument listed in the format string format. These routines are compiler dependent due to the nature of compiler differences in processing functions and operands.

Function **strchr** () checks for existence of a string by checking the value of pointer *s* . Function **find_arg_type** () parses a format string of the type passed to any formatted output from printf, fprintf, or sprintf. It then returns an appropriate argument type *ARG_type* depending on the value passed to it.

Utility function **format_decoder** () is the top level routine in this program which does the equivalent of **sprintf**, placing a formatted string in a character array. Input arguments to **format_decoder** () are a pointer to an allocated buffer, a format string and a pointer to a list of arguments (as initialized by *va_start* (list). Why not use **sprintf** ? Because it WILL NOT accept a pointer to an argument list. Therefore, it can't be called by a routine that has arguments passed to it from a variable argument list such as timed_printf or error_printf.

Before this utility was provided, a funciton called **_doprnt** was used. Since this is NOT guaranteed to exist in any C implementation, a compiler independent and machine independent function is required to satisfy portability requirements.

Function **copybuf** copies string characters between two pointers (start and end) to output buffer *buf* .

Includes:

"stdio.h"
"strings.h"
"ctype.h"
"varargs.h"

Defines:

ARG_NONE
ARG_CHAR
ARG_POINTER
ARG_INT
ARG_LONG
ARG_FLOAT
ARG_DOUBLE

2.6.4.15.1 strchr

Function **strchr** () checks for the existence of a string, *c*, by checking the value of pointer *s*. If *s* is not zero (false), it then returns the contents of the string *s*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
s	pointer to char	Standard
c	char	Standard
Return Values		
Return Value	**Type**	**Meaning**
s	static pointer to char	the contents of the string
0	static pointer to char	string is \alse

Table 2.6-120: strchr Information.

2.6.4.15.2 find_arg_type

Function **find_arg_type** () parses a format string of the type passed to any formatted output from printf, fprintf, or sprintf. It then returns an appropriate argument type *ARG_type* depending on the value passed to it.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
s	pointer to char	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
fp	pointer to char	Standard
strchr()	pointer to extern char	Standard
big	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
ARG_NONE	static int	default
ARG_POINTER	static int	a pointer was passed
ARG_CHAR	static int	a char was passed
ARG_LONG	static int	a long was passed
ARG_INT	static int	an int was passed
ARG_DOUBLE	static int	a double was passed
ARG_FLOAT	static int	a float was passed
Calls		
Function	**Where Described**	
strchr	Section 2.6.4.15.1	

Table 2.6-121: find_arg_type Information.

2.6.4.15.3 format_decoder

This is the top level routine for this utility. Given a pointer to a buffer (that you have allocated), a format string and a pointer to a list of args (as initialized by va_start) this routine will do the equivalent of an **sprintf**.

Defines:
WORKSPACE_SIZE

Parameters		
Parameter	**Type**	**Where Typedef Declared**
buf_addr	pointer to char	Standard
fmt	pointer to char	Standard
Internal and External Variables		
Variable	**Type**	**Where Typedef Declared**
start_next_fmt	pointer to char	Standard
end_next_fmt	pointer to char	Standard
workspace	pointer to char	Standard
str_size	int	Standard
done	int	Standard
arg_type	int	Standard
int_arg	int	Standard
double_arg	double	Standard
long_arg	long	Standard
pointer_arg	pointer to char	Standard
copybuf()	static void	Standard
strchr()	pointer to extern char	Standard
malloc	pointer to extern char	Standard
Return Values		
Return Value	**Type**	**Meaning**
str_size	int	size of buffer
Calls		
Function	**Where Described**	
strchrt	Section 2.6.4.15.1	
find_arg_type	Section 2.6.4.15.2	

Table 2.6-122: format_decoder Information.

2.6.4.15.4 copybuf

This function copies string characters between two pointers (*start* and *end*) to output buffer *buf* .

Parameters		
Parameter	**Type**	**Where Typedef Declared**
buf	pointer to char	Standard
start	pointer to char	Standard
end	pointer to char	Standard

Table 2.6-123: copybuf Information.

2.6.4.16 libutil.h

(/simnet/release/src/libsrc/libutil/libutil.h)

The following routines are declared to be external:

dump_core()
sin_cos_to_deg()
add_dead_zone()

2.6.4.17 pr_R4P3D.c

(/simnet/release/src/libsrc/libutil/pr_R4P3D.c)

This file contains one routine, **print_R4P3D()**, which prints an R4P3D.

The following are included:

"stdio.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"

2.6.4.17.1 print_R4P3D

This routine prints an R4P3D matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rp	pointer to R4P3D	dgi_stdg.h
s	char	Standard

Table 2.6-124: print_R4P3D Information.

2.6.4.18 pr_TF1.c
(/simnet/release/src/libsrc/libutil/pr_TF1.c)

This file contains one procedure, **print_TF1()**, which prints a TF1 matrix.

The following are included:
"stdio.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"

2.6.4.18.1 print_TF1

This procedure prints a TF1 matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tf	pointer to TF1	sim_cig_if.h
s	char	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
j	int	Standard

Table 2.6-125: print_TF1 Information.

2.6.4.19 pr_TF2.c
(/simnet/release/src/libsrc/libutil/pr_TF2.c)

This file contains one routine, **print_TF2()**, which prints a TF2 matrix.

The following are included:
"stdio.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"

2.6.4.19.1 print_TF2

This routine prints a TF2 matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tf	pointer to TF2	sim_cig_if.h
s	char	Standard

Table 2.6-126: print_TF2 Information.

2.6.4.21 strtok.c

(/simnet/release/src/libsrc/libutil/strtok.c)

This file provides the functionality of the standard Berkeley C file strtok.c. It was written because the Butterfly does not run with Berkeley C. These routines are not defined for the Masscomp.

The following are declared:

***strtok()**
***strtok_skip()**
***strtok_find()**

2.6.4.21.1 strtok_skip

This routine examines a string.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
str	pointer to char	Standard
col	pointer to char	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
p	pointer to char	Standard
q	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
p	pointer to char	ran out of column chars
((char)0)	pointer to char	all matched

Table 2.6-127: strtok_skip Information.

2.6.4.21.2 strtok_find

This routine finds a string specified by *str*, given a token to parse on (*col*). It returns an indication of success or failure.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
str	pointer to char	Standard
col	pointer to char	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
p	pointer to char	Standard
q	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
p	pointer to char	matched
((char)0)	pointer to char	no match

Table 2.6-128: strtok_find Information.

2.6.4.21.3 strtok

This routine parses a string and passes it back to the user. It contains the same functionality of the standard C call **strtok**.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
str	pointer to char	Standard
col	pointer to char	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
p	pointer to char	Standard
temp_p	pointer to char	Standard
q	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
((char)0)	pointer to char	the string is logically equal to 0
tmp_p	pointer to char	parsed string
str	pointer to char	parsed string
Calls		
Function	**Where Described**	
strtok_skip	Section 2.6.4.21.1	
strtok_find	Section 2.6.4.21.2	

Table 2.6-129: strtok Information.

2.6.4.22 t_mat_dump.c
(/simnet/release/src/libsrc/libutil/t_mat_dump.c)

This file contains one routine, **timed_mat_dump**, which dumps a matrix to the standard output at a specified time.

2.6.4.22.1 timed_mat_dump

This routine dumps a matrix to the standard output at a specified time.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
str	char	Standard
mat	T_MATRIX	sim_types.h
Calls		
Function	**Where Described**	
timed_printf	Section 2.6.4.24.1	

Table 2.6-130: timed_mat_dump Information.

2.6.4.24 t_vec_dump.c
(/simnet/release/src/libsrc/libutil/t_vec_dump.c)

This file contains one routine, **timed_vec_dump**, which dumps a vector to the standard output at a specified time.

2.6.4.23.1 timed_vec_dump

This routine dumps a vector to the standard output at a specified time.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
str	char	Standard
v	VECTOR	sim_types.h
Calls		
Function	**Where Described**	
timed_printf	Section 2.6.4.24.1	

Table 2.6-131: timed_vec_dump Information.

2.6.4.24 timed_printf.c
(/simnet/release/src/libsrc/libutil [libutil])

This file contains a routine which prints out at a specified time.

Includes:
"stdio.h"
"timers.h"

The following is declared:
interval

2.6.4.24.1 timed_printf

This routine prints out every *interval* ticks. *ctl* is the printout control, and *args[]* is the list of arguments.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ctl	pointer to char	Standard
args[]	int	Standard
Calls		
Function	**Where Described**	
timers_get_current_tick	Section 2.6.3.1.1	

Table 2.6-132: timed_printf Information.

2.6.4.24.2 timed_printf_set

This routine sets the printing interval.

Parameter	**Type**	**Where Typedef Declared**
val	int	Standard

Table 2.6-133: timed_printf_set Information.

2.6.5 libshm

(/simnet/release/src/libsrc/libshm [libshm])

Libshm provides facilities for mapping in regions of shared memory. It encapsulates a sequence of system calls into a single uniform interface. Shared memory segments are used in the SIMNET simulations to provide shared access by (potentially) multiple processes to memory buffers used to communicate with I/O devices.

A shared memory segment is attached to by calling **attachshm()**, which takes as arguments a key (identifier), segment size in bytes, and a flag that specifies whether or not to create the segment. A process can detach from a segment by calling **detachshm()**. A shared memory segment may be removed by calling **removeshm()**.

2.6.5.1 attach.c

(/simnet/release/src/libsrc/libshm/attach.c)

This file comtains one procedure, **attachshm**, which attaches to a shared memory segment.

Includes:

"stdio.h'
"fcntl.h"
"errno.h"
"sys/types.h:"
"sys/pc.h"
"sys/shm.h"
"sys/sem.h"
"signal.h"
"shmcontrol.h"

2.6.5.1.1 attachshm

This routine is called to attach the shared memory associated with the *key* (of size *size* bytes) to the calling process. The shared segment is created if the *createflag* is TRUE and the segment doesn't exist. If the flag is FALSE and the segment doesn't exist, this routine fails and returns a null pointer.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
key	int	Standard
size	int	Standard
createflag	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
sbrk()	pointer to char	Standard
cp	pointer to char	Standard
id	int	Standard
shmat	pointer to char	Standard
p	pointer to char	Standard
i	int	Standard
flags	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
NULL	char	routine failed
p	pointer to char	address of shared memory segment that you are attaching to

Table 2.6-134: attachshm Information.

2.6.5.2 detach.c

(/simnet/release/src/libsrc/libshm/detach.c)

This file comtains one procedure, **detachshm**, which detaches from the shared memory segment.

Includes:

"stdio.h'
"fcntl.h"
"errno.h"
"sys/types.h:"
"sys/pc.h"
"sys/shm.h"
"sys/sem.h"
"signal.h"
"shmcontrol.h"

2.6.5.2.1 detachshm

This routine detaches the shared memory at the argument address from the calling process. It returns 0 if successful and -1 otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
shmaddr	pointer to char	Standard
Return Values		
Return Value	**Type**	**Meaning**
-1	int	routine failed
0	int	routine succeeded

Table 2.6-135: detachshm Information.

2.6.5.3 remove.c

(/simnet/release/src/libsrc/libshm/remove.c)

This file contains one procedure, **removeshm**, which removes the shared memory segment.

Includes:

"stdio.h'
"fcntl.h"
"errno.h"
"sys/types.h:"
"sys/pc.h"
"sys/shm.h"
"sys/sem.h"
"signal.h"
"shmcontrol.h"

2.6.5.3.1 removeshm

This routine removes the shared memoruy specified by the key from the calling process. It returns 0 if successful and -1 otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
key	int	Standard
size	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
id	int	Standard
flags	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
-1	int	routine failed
0	int	routine succeeded

Table 2.6-136: removeshm Information.

2.6.5.4 shmcontrol.h

(/common/libsource/libshm/shmcontrol.h)

The following are declared for use outside of libshm:

attachshm()
detachshm()
removeshm()

2.6.6 libmove

(./simnet/release/src/libsrc/libmove [libmove])

libmove contains utilities for moving data. It exists to support moving data to/from I/O devices that place restrictions on the types of access supported and to provide performance enhancements for operating system and hardware platforms whose native facilities were found to be suboptimal. These routines are written in assembly language.

2.6.7 libser

(./simnet/release/src/libsrc/libser [libser])

libser provides access to the heartbeat function of the HPSM serial card used to interface to the IDC boards. The heartbeat is periodically checked to detect whether the card has failed.

2.6.7.1 m1_mem_dfn.h

(./simnet/release/src/libsrc/libser/m1_mem_dfn.h)

A number of variables are declared as external for use in libshm.

2.6.7.2 ser_status.c

(./simnet/release/src/libsrc/libser/ser_status.c)

This file contains functions which are used to determine if the HPSM serial card has failed. **Ser_heartbeat_init** is called to initialize access to the heartbeat location, while **ser_heartbeat** is called to obtain the heartbeat value.

heartbeat is declared as static short.

2.6.7.2.1 ser_heartbeat

This routine is called to obtain the heartbeat value. It returns TRUE if the current value of the location in host memory that is continually incremented by the HPSM card software has changed since the last call to this routine. It returns FALSE otherwise. This routine should not be called more often than about once per second to make sure that the HPSM card has had time to increment the status word. It shoud be called at least once each minute to minimize the possibility of missing a change due to roll over.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
new_heartbeat	short	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	current value in location of shared memory has changed
FALSE	int	current value in location of shared memory has not changed

Table 2.6-137: ser_heartbeat Information.

2.6.7.2.2 ser_heartbeat_init

This routine is initialized for monitoring **ser_heartbeat()**. It must be called before calling **ser_heartbeat()**. *heartbeat* is initialized to *pser_heartbeat*.

2.6.8 libfifo
(./simnet/release/src/libsrc/libfifo [libfifo])

libfifo provides an interface for sending output to a serial device. It allows the user to queue up to 128 messages, each with a length of up to 10 bytes. Routines are provided to create a queue, enqueue a message, and send all messages currently on the queue to a serial device. This interface is used by libsound (see section 2.1.3.1) and libidc (see section 2.1.4.1.1).

2.6.8.1 f_dequeue.c
(./simnet/release/src/libsrc/libfifo/f_dequeue.c)

This file contains a routine which moves messages along the message queue.

Includes:
"stdio.h"
"fcntl.h"
"sim_dfns.h"
"fifo_dfn.h"
"fifo.h"

Defines:
FIFO_DEBUG

2.6.8.1.1 fifo_dequeue

This routine removes messages from the message queue. If the queue is not empty, the length of the current message is returned, and the pointer is moved to the next message. If the queue is empty, this routine returns a 0.

Parameters		
Parameter	Type	Where Typedef Declared
fifop	pointer to FIFO	fifo_dfn.h
bufp	pointer to char	Standard
Internal Variables		
Internal Variable	Type	Where Typedef Declared
length	int	Standard
Return Values		
Return Value	Type	Meaning
0	int	the queue is empty
length	int	the length of the buffer
Calls		
Function	Where Described	
FIFO_EMPTY	Section 2.6.8.8	
NEXT_OUT	Section 2.6.8.8	

Table 2.6-138: fifo_dequeue Information.

2.6.8.2 f_enqueue.c

(./simnet/release/src/libsrc/libfifo/f_enqueue.c)

This file contains a routine which sends a message to the message queue.

Includes:
"stdio.h"
"fcntl.h"
"sim_dfns.h"
"fifo_dfn.h"
"fifo.h"

2.6.8.2.1 fifo_enqueue

This routine sends a message to the message queue, given the message string, the string length, and the pointer to the FIFO port that the message is to be sent to. This routine is called by output producing routines.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fifop	pointer to FIFO	fifo_dfn.h
string	pointer to char	Standard
length	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
sccptr	char	Standard
Return Values		
Return Value	**Type**	**Meaning**
FALSE	boolean	full fifo port fifo message too long
TRUE	boolean	message sent to output port
Calls		
Function	**Where Described**	
FIFO_FULL(fifop)	Section 2.6.8.8	
NEXT_IN(fifop)	Section 2.6.8.8	

Table 2.6-139: fifo_enqueue Information.

2.6.8.3 f_init.c
(./simnet/release/src/libsrc/libfifo/f_init.c)

This file contains a routine which initializes the fifo interface.

2.6.8.3.1 fifo_init

This routine initializes the FIFO segment of shared memory, given a pointer to the FIFO array and a port number This routine is called by processes that initializes shared memory.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fifop	pointer to FIFO	fifo_dfn.h
port_number	int	Standard

Table 2.6-140: fifo_init Information.

2.6.8.3.2 fifo_uninit

This routine is not defined for a Masscomp or Butterfly machine.

2.6.8.4 f_open_out.c
(./simnet/release/src/libsrc/libfifo/f_open/c)

The following files are included:
"stdio.h"
"strings.h"
"fcntl.h"
"sim_dfns.h"
"fifo_dfn.h"
"fifo.h"

The following are defined:
FIFO_DEBUG

2.6.8.4.1 open_up_output_port

This routine opens an output port specified by fifop. It is similar to a UNIX open.

If a Butterfly machine is used:

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fifop	pointer to FIFO	fifo_dfn.h
Return Values		
Return Value	**Type**	**Meaning**
FALSE	boolean	couldn't open the output port

Table 2.6-141: open_up_output_port Information for the Butterfly.

If a Masscomp machine is used:

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fifop	pointer to FIFO	fifo_dfn.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
buf[80]	char	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	boolean	fifo port opened
FALSE	boolean	couldn't open fifo port

Table 2.6-142: open_up_output_port Information for the Masscomp.

2.6.8.4.2 close_output_port

This routine isn't used by the Masscomp or Butterfly.

2.6.8.5 f_print.c

(./simnet/release/src/libsrc/libfifo/f_print.c)

This file contains a routine which prints information about the fifo interface.

The following are included:

"stdio.h"
"fcntl.h"
"sim_dfns.h"
"fifo_dfn.h"
"fifo.h"

2.6.8.5.1 fifo_print

This routine prints information about the fifo interface. It is used as a debugging tool.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fifop	pointer to FIFO	fifo_dfn.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard

Table 2.6-143: fifo_print Information.

2.6.8.6 f_send_out.c
(./simnet/release/src/libsrc/libfifo/f_send_out.d)

This file contains a routine which sends message strings to an output port.

Includes:
"stdio.h"
"fcntl.h"
"sim_dfns.h"
"fifo_dfn.h"
"fifo.h"

2.6.8.6.1 send_output_to_port

This routine repeatedly calls fifo_dequeue and takes the message in the queue and writes it to the output port. If the message is sent, the routine returns TRUE , and if no message is sent, the routine returns FALSE. This routine is not used by the Butterfly.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fifop	pointer to FIFO	fifo_dfn.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
length	int	Standard
buf[MESSAGE_SIZE]	char	Standard
wrote_something	boolean	Standard
Return Values		
Return Value	**Type**	**Meaning**
wrote_something	boolean	if TRUE, wrote message to output port if FALSE, message not sent
Calls		
Function	**Where Described**	
fifo_dequeue	Section 2.6.8.1.1	

Table 2.6-144: send_output_to_port Information.

2.6.8.7 fifo.h
(./simnet/release/src/libsrc/libfifo/fifo.h)

This file declares the following procedures to be external:

fifo_init()
fifo_print()
fifo_enqueue()
fifo_dequeue()
open_up_output_port()
send_output_to_port()

2.6.8.8 fifo_dfn.h
(./simnet/release/src/libsrc/libfifo/fifo_dfn.h)

The following macros are defined:
NEXT_IN
NEXT_OUT
FIFO_EMPTY
FIFO_FULL

The FIFO structure is defined.

2.6.9 libevent
(./simnet/release/src/libsrc/libevent [libevent])

Event IDs are used to distinguish packets that are sent out on the network. Routines for generating a unique event ID is contained in libevent.

2.6.9.1 event.c
(./simnet/release/src/libsrc/libevent/event.c)

This file contains routines which generate a unique event ID.

Includes:

"stdio.h"
"math.h"
"sim_drfn.h"
"sim_macros.h"
"types.h"

The following is declared:

eventy_counter

2.6.9.1.1 event_init_eventid

This procedure initializes the event counter. *new_event_cnt* is the event ID to start numbering from.

Parameters		
Parameter	Type	Where Typedef Declared
new_event_cnt	long int	Standard

Table 2.6-145: event_init_eventid Information.

2.6.9.1.2 event_get_eventid

This procedure sequences to the next event. *skip* indicates the number of events to be skipped.

Parameters		
Parameter	Type	Where Typedef Declared
skip	int	Standard
Return Values		
Return Value	Type	Meaning
event_counter	long int	the id of the next event

Table 2.6-146: event_get_eventid Information.

2.6.9.2 libevent.h
(./simnet/release/src/libsrc/libevent/libevent.h)

The following are defined:

SKIP
NO_SKIP
NO_EVENT
NO_AGENT

The following are declared to be external:

event_init_eventid()
event_get_eventid()

2.6.10 libveh

(./simnet/release/src/vehicle/libsrc/libveh [libveh])

Libveh provides a number of functions which return information about other vehicles. The available information includes the type of vehicle, its role on the battlefield and which side of the battle it is on. Libveh provides the ability to set the force of the simulated vehicle for comparison with other vehicles. Libveh is the only CSU required to perform these tasks.

2.6.10.1 is_air_veh.c

(./simnet/release/src/vehicle/libsrc/libveh/is_air_veh.c)

This file contains a routine which determines if the vehicle in question is an air vehicle.

Includes:

"sim_dfns.h"
"basic.h"
"obj_type.h"

2.6.10.1.1 is_air_vehicle

This routine returns TRUE if the specified vehicle is an air vehicle. Otherwise, it returns FALSE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	ObjectType	p_sim.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	vehicle is an air vehicle
FALSE	int	vehicle is not an air vehicle

Table 2.6-147: is_air_vehicle Information.

2.6.10.2 is_ammo_veh.c
(./simnet/release/src/vehicle/libsrc/libveh /is_ammo_veh.c)

This file contains routines which determine if the vehicle in question is an ammunition carrier.

Includes:
"sim_dfns.h"
"pro_sim.h"
"obj_type.h"
"veh_type.h"

2.6.10.2.1 is_ammo_vehicle

This routine indicates whether or not a vehicle is an ammunition carrier. It returns a pointer to the vehicle appearance variant.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to VehicleAppearanceVariant	p_sim.h
Return Values		
Return Value	**Type**	**Meaning**
pkt-> capabilities.ammunitionSupply	int	indication of whether or not the vehicle carries ammunition. 1 means the vehicle carries ammo 0 means the vehicle doesn't carry ammo

Table 2.6-148: is_ammo_vehicle Information.

2.6.10.2.2 is_ammo_carrier

This routine determines if the vehicle in question is an American ammunition carrier. It returns TRUE if this is the case and returns FALSE otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	ObjectType	p_sim.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	the vehicle is an American ammunition carrier
FALSE	int	the vehicle is not an American ammunition carrier

Table 2.6-149: is_ammo_carrier Information.

2.6.10.3 is_anti_air.c
(./simnet/release/src/vehicle/libsrc/libveh/is_anti_air.c)

This file contains a routine which determines if the vehicle in question is an antiaircraft vehicle.

Includes:
"sim_dfns.h"
"basic.h"
"obj_type.h"

2.6.10.3.1 is_anti_aircraft

This routine returns TRUE if the specified vehicle is an antiaircraft vehicle. Otherwise, it returns FALSE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	ObjectType	p_sim.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	vehicle is an antiaircraft vehicle
FALSE	int	vehicle is not an antiaircraft vehicle

Table 2.6-150: is_anti_aircraft Information.

2.6.10.4 is_apc.c
(./simnet/release/src/vehicle/libsrc/libveh/is_apc.c)

This file contains a routine which determines if the vehicle in question is an armored personnel carrier.

Includes:
"sim_dfns.h"
"basic.h"
"obj_type.h"

2.6.10.4.1 is_personnel_carrier

This routine returns TRUE if the specified vehicle is an armored personnel carrier. Otherwise, it returns FALSE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	ObjectType	p_sim.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	vehicle is an armored personnel carrier
FALSE	int	vehicle is not an armored personnel carrier

Table 2.6-151: is_personnel_carrier Information.

2.6.10.5 is_att_rwa.c
(./simnet/release/src/vehicle/libsrc/libveh/is_att_rwa.c)

This file contains a routine which determines if the vehicle in question is an attack rotary wing aircraft (helicopter).

Includes:
"sim_dfns.h"
"basic.h"
"obj_type.h"

2.6.10.5.1 is_attack_rwa

This routine returns TRUE if the specified vehicle is an attack rotary wing aircraft (helicopter). Otherwise, it returns FALSE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	ObjectType	p_sim.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	vehicle is an attack rotary wing aircraft
FALSE	int	vehicle is not an attack rotary wing aircraft

Table 2.6-152: is_attack_rwa Information.

2.6.10.6 is_friend.c
(./simnet/release/src/vehicle/libsrc/libveh/is_friend.c)

This file contains routines which determine if a vehicle is friendly, set a vehicle's force id, and returns a vehicle's force id.

Includes:
stdio.h"
"sim_types.h"
"sim_dfns.h"
"pro_sim.h"
"veh_type.h"

The variable *our_force* is declared.

2.6.10.6.1 is_friendly

This routine determines if a vehicle is friendly, that is, if a vehicle is on your side or if it is a target vehicle. *his_force* is the force id. If the vehicle is friendly, the routine returns TRUE. Otherwise, it returns FALSE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
his_force	ForceID	basic.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	vehicle is friendly
FALSE	int	vehicle is not friendly

Table 2.6-153: is_friendly Information.

2.6.10.6.2 veh_set_force

This routine changes the force id of a vehicle. *new_force* is the new force id of the vehicle.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
new_force	ForceID	basic.h

Table 2.6-154: veh_set_force Information.

2.6.10.6.3 veh_get_force

This routine returns the force id of our vehicle.

Return Values		
Return Value	**Type**	**Meaning**
our_force	ForceID	our force id

Table 2.6-155: veh_get_force Information.

2.6.10.7 is_fuel_veh.c
(./simnet/release/src/vehicle/libsrc/libveh/is_fuel.c)

This file contains a routine which determines if a vehicle is a fuel supply truck.

Includes:
"sim_dfns.h"
"pro_sim.h"

2.6.10.7.1 is_fuel_vehicle

This routine returns a pointer to the vehicle appearance variant to indicate whether or not a vehicle is a fuel supply truck.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to VehicleAppearanceVariant	p_sim.h
Return Values		
Return Value	**Type**	**Meaning**
pkt-> capabilities.FuelSupply	int	indicates if vehicle carries fuel. 1 means it is a fuel truck 0 means it is not a fuel truck

Table 2.6-156: is_fuel_vehicle Information.

2.6.10.8 is_fwa.c
(./simnet/release/src/vehicle/libsrc/libveh/is_fwa.c)

This file contains a routine which determines if the vehicle in question is fixed wing aircraft.

Includes:
"sim_dfns.h"
"basic.h"
"obj_type.h"

2.6.10.8.1 is_fwa

This routine returns TRUE if the specified vehicle is a fixed wing aircraft. Otherwise, it returns FALSE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	ObjectType	p_sim.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	vehicle is a fixed wing aircraft
FALSE	int	vehicle is not a fixed wing aircraft

Table 2.6-157: is_fwa Information.

2.6.10.9 is_mb_tank.c
(./simnet/release/src/vehicle/libsrc/libveh/is_mb_tank.c)

This file contains a routine which determines if the vehicle in question is a main battle tank.

Includes:
"sim_dfns.h"
"basic.h"
"obj_type.h"

2.6.10.1.1 is_air_vehicle

This routine returns TRUE if the specified vehicle is a main battle tank. Otherwise, it returns FALSE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	ObjectType	p_sim.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	vehicle is a main battle tank
FALSE	int	vehicle is not a main battle tank

Table 2.6-158: is_main_battle_tank Information.

2.6.10.10 is_rep_veh.c
(./simnet/release/src/vehicle/libsrc/libveh /is_rep_veh.c)

This file contains a routine which determines if a vehicle is a repair vehicle.

Includes:
"sim_dfns.h"
"pro_sim.h"

2.6.10.10.1 is_repair_vehicle

This routine returns a pointer to the vehicle appearance variant to indicate if the vehicle in question is a repair vehicle.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to VehicleAppearanceVariant	p_sim.h

Return Values		
Return Value	**Type**	**Meaning**
pkt-> capabilities.repair	int	indicates if vehicle is a repair vehicle 0 means the vehicle is not a repair vehicle 1 means the vehicle is a repair vehicle

Table 2.6-159: is_repair_vehicle Information.

2.6.10.11 is_rwa.c
(./simnet/release/src/vehicle/libsrc/libveh/is_rwa.c)

This file contains a routine which determines if the vehicle in question is a rotary wing aircraft.

Includes:
"sim_dfns.h"
"basic.h"
"obj_type.h"

2.6.10.11.1 is_rwa

This routine returns TRUE if the specified vehicle is a rotary wing aircraft. Otherwise, it returns FALSE.

Parameters		
Parameter	Type	Where Typedef Declared
type	ObjectType	p_sim.h

Return Values		
Return Value	Type	Meaning
TRUE	int	vehicle is a rotary wing aircraft
FALSE	int	vehicle is not a rotary wing aircraft

Table 2.6-160: is_rwa Information.

2.6.10.12 libveh.h
(./simnet/release/src/vehicle/libsrc/libveh/libveh.h)

This file contains declarations of the following functions which are used in "libveh":

is_ammo_vehicle()
is_fuel_vehicle()
is_repair_vehicle()
is_main_battle_tank()
is_personnel_carrier()
is_anti_aircraft()
is_air_vehicle()
is_priority_vehicle()
is_attack_rwa()
is_fwa()
map_othervehs_net_id_to_mycig_id()

2.6.11 libmap
(simnet/release/src/libsrc/libmap [libmap])

The dynamic models and effects displayed by the cig are contained in the dynamic element database (DED). There are 256 possible models and 256 possible effects. Each model/effect is assigned an 8-bit index from 0-255. The assignment of model/effect indices is specific to a particular DED.

The SIMNET protocols use a method of specifying models and ammunition types which is independent of that of the cig. It uses a 32-bit integer where adjacent groups of bits contain hierarchical information about the object being described.

The library, "libmap," contains routines which map model and effect definitions from the SIMNET protocol description to the cig index and visa versa. In addition, it also maps information specified by the protocol appearance field to bits defined in the cig interface to describe enhancers (Application Specific Identifiers or ASIDs) for the models. These include features such as dust clouds, flaming, and smoking. The ASID field in the cig interface also allows one to specify a 3 character bumper number for which each character can be 0-9, A-D, H, or a blank.

"Libmap" reads in 3 files, one which describes the ASID bits, one which indicates the mapping from SIMNET protocol vehicles to cig models and one which indicates the mapping for SIMNET protocol ammunition types to their corresponding effects as well as damage files. The file names are specified in a parameter file which is read in during startup.

2.6.11.1 damage.c
(./simnet/release/src/libsrc/libmap/damage.c)

This file contains routines which check to see if the appropriate damage files exist. If they do exist, the damage files are read in. For each vehicle, damage files exist for various ammunition types.

The following files are included:

"stdio.h"
"signal.h'
"ctype.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libmap_dfn.h"
"libmap.h"
"libfail.h"

The following is defined:

NO_DAMAGE_TABLE_INDEX

The following are declared externally:

ded_map_entries[]
number_of_entries
last_in_ammo_class[]
number_of_damage_files
check_for_nonexistant_damage_files()

2.6.11.1.1 map_get_damage_files

This ıoutine reads in damage files. The damage file name is checked to see if it is recognized as one that has already been read in. If the file has been read previously, the index is made to point to the already read in damage file. If the damage file hasn't been read in yet, then it is read in. If the file is read in successfully, the index is made to point to the correct damage file. If the damage file was not available, this entry is marked so it can be updated to point to the default damage table.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
j	int	Standard
number_of_damage_files	int	Standard
alreadsy_read_this_damage_file	int	Standard
Calls		
Function	**Where Described**	
cfail_read_damage_file	Section 2.5.4.5.2	
check_for_nonexistant_damage_files	Section 2.6.11.1.2	

Table 2.6-161: map_get_damage_files Information.

2.6.11.1.2 check_for_nonexistant_damage_files

This routine checks for nonexistant damage files. Default entries must have valid damage tables. If the damage table can not be found (not on disk) for a given entry, then the damage file index is made to be the same as the default entry. Update *first_in_ammo_class* to be *first_in_class (i+1)*.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
j	int	Standard
first_in_ammo_class	int	Standard

Table 2.6-162: check_for_nonexistent_damage_files Information.

2.6.11.2 get_entry.c

(./simnet/release/src/libsrc/libmap/get_entry.c)

This file contains routines used to map between the network and CIG.

The following files are included:

- "stdio.h"
- "basic.h"
- "obj_type.h"
- "libmap_dfn.h"

The following constants are defined:

- CALIBER_SMALL
- CALIBER_MEDIUM

The following are declared externally:

- ded_map_entries[]
- last_in_ammo_class[]
- number_of_entries
- **search_obj_types**()

2.6.11.2.1 map_get_ammo_entry_from_network_type

This routine maps ammo types from the network to the CIG. The ammunition type is represented by *ammo_type*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_type	ObjectType	p_sim.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
caliber	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
search_obj_types(ammo_type, MAP_MISSILES)	int	missiles
search_obj_types(ammo_type, MAP_PROJ_SMALL)	int	small projectiles
search_obj_types(ammo_type, MAP_PROJ_MEDIUM)	int	medium projectiles
search_obj_types(ammo_type, MAP_PROJ_LARGE)	int	large projectiles
search_obj_types(ammo_type, MAP_BOMBS)	int	mines are treated as bombs
-1	int	default
Calls		
Function	**Where Described**	
search_obj_types	Section 2.6.11.2.2	

Table 2.6-163: map_get_ammo_entry_from_network_type Information.

2.6.11.2.2 search_obj_types

Given an object type, the routine searches through the data structure and returns the low integer that represents that object type.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
last_in_ammo_class_index	int	Standard
ammo_type	ObjectType	p_sim.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
start_index	int	Standard
end_index	int	Standard
i	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
i	int	index corresponding to object type
end_index	int	last entry in table

Table 2.6-164: search_obj_types Information.

2.6.11.2.3 map_get_network_type_from_ammo_entry

This routine converts the index in the map array into a network munition. *ammo_entry* is the entry in the map array which represents a particular ammunition type.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
objectIrrelevant	ObjectType	irrelevant object
ded_map_entries [ammo_entry] -> ammunition	ObjectType	ammunition type

Table 2.6-165: map_get_network_type_from_ammo_entry Information.

2.6.11.2.4 map_get_burst_ground_from_ammo_entry

This routine returns the low integer that represents the ground burst effect for a particular ammunition type. ammo_entry is the member of the map array which represents a

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
dont_know	unsigned char	don't recognize ammunition type
ded_map_entries[ammo_entry] -> burst_ground	unsigned char	ground burst effect for an ammunition type

Table 2.6-166: map_get_burst_ground_from_ammo_entry Information.

2.6.11.2.5 map_get_burst_air_from_ammo_entry

This routine returns the low integer that represents the air burst effect for a particular ammunition type.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
dont_know	unsigned char	don' recognize ammunition type
ded_map_entries [ammo_entry] -> burst_air	unsigned char	air burst effect for ammunition type

Table 2.6-167: map_get_burst_air_from_ammo_entry Information.

2.6.11.2.6 map_get_burst_armor_from_ammo_entry

This routine returns the low integer that represents the armor burst effect for a particular ammunition type.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
dont_know	unsigned char	don't recognize ammunition type
ded_map_entries [ammo_entry] -> burst_armor	unsigned char	armor burst effect for ammunition type

Table 2.6-168: map_get_burst_armor_from_ammo_entry Information.

2.6.11.2.7 map_get_burst_wood_from_ammo_entry

This routine returns the low integer that represents the wood burst effect for a particular ammunition type.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
dont_know	unsigned char	don't recognize ammunition type
ded_map_entries [ammo_entry] -> burst_wood	unsigned char	wood burst effect for ammunition type

Table 2.6-169: map_get_burst_wood_from_ammo_entry Information.

2.6.11.2.8 map_get_burst_other_from_ammo_entry

This routine returns the low integer that represents the burst effect for unknown objects given a particular type of ammunition.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
dont_know	unsigned char	don't recognize ammunition type
ded_map_entries [ammo_entry] -> burst_other	unsigned char	burst effect for ammunition type

Table 2.6-170: map_get_burst_other_from_ammo_entry Information.

2.6.11.2.9 map_get_tracer_from_ammo_entry

This routine returns the low integer that represents a tracer for a given projectile.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
dont_know	unsigned char	no effect should be shown
ded_map_entries [ammo_entry] -> tracer	unsigned char	tracer effect based on projectile type

Table 2.6-171: map_get_tracer_from_ammo_entry Information.

2.6.11.2.10 map_get_muzzle_flash_me_from_ammo_entry

This routine returns the low integer that represents the own muzzle flash effect for a given ammunition type.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
dont_know	unsigned char	ammo type not recognized
ded_map_entries [ammo_entry] -> muzzle_flash_me	unsigned char	muzzle flash effect for a given ammunition type

Table 2.6-172: map_get_muzzle_flash_me_from_ammo_entry Information.

2.6.11.2.11 map_get_muzzle_flash_other_from_ammo_entry

This routine the low integer that represents the muzzle flash effects on another vehicle given a particular ammunition type.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
dont_know	unsigned char	ammunition type not recognized
ded_map_entries [ammo_entry] -> muzzle_flash_other	unsigned char	other vehicle's muzzle flash effect for ammunition type

Table 2.6-173: map_get_muzzle_flash_other_from_ammo_entry Information.

2.6.11.2.12 map_get_damage_file_index_from_ammo_entry

This routine returns the index into the array of damage tables for a given ammunition type.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
-1	int	ammunition type is not recognized
ded_map_entries [ammo_entry] -> damage_file_index	int	index into damage file array

Table 2.6-174: map_get_damage_file_index_from_ammo_entry Information.

2.6.11.2.13 map_get_ammo_class_from_ammo_entry

This routine returns the class of ammunition given the low integer that represents the ammunition type.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
-1	int	don't recognize the ammunition type
MAP_BOMBS	int	the class is bomb
MAP_MISSILES	int	the class is missile
MAP_PROJ_SMALL	int	the class is small projectile
MAP_PROJ_MEDIUM	int	the class is medium projectile
MAP_PROJ_LARGE	int	the class is large projectile

Table 2.6-175: map_get_ammo_class_from_ammo_entry Information.

2.6.11.2.14 map_is_bomb

Given a list of items, this routine returns TRUE if an item is a bomb and returns FALSE otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
(ammo_entry>=0)&& (ammo_entry<= last_in_ammo_class [MAP_BOMBS])	int	TRUE if it is a bomb FALSE if not a bomb

Table 2.6-176: map_is_bomb Information.

2.6.11.2.15 map_is_missile

Given a list of items, this routine returns TRUE if an item is a missile and returns FALSE otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
(ammo_entry>last_in_ammo_class [MAP_BOMBS)) && (ammo_entry<= last_in_ammo_class[MAP_MISSILES])	int	TRUE if the item is a missile FALSE if the item isn't a missile

Table 2.6-177: map_is_missile Information.

2.6.11.2.16 map_is_projectile

Given a list of items, this routine returns TRUE if an item is a projectile and returns FALSE otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_entry	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
(ammo_entry>last_in_ammo_class [MAP_MISSILES)) && (ammo_entry<= last_in_ammo_class[MAP_PROJ_LARGE])	unsigned char	TRUE if it is a projectile FALSE if it is not a projectile

Table 2.6-178: map_is_projectile Information.

2.6.11.3 map_ammo.c
(./simnet/release/src/libsrc/libmap/map_ammo.c)

This file contains the routines which read a map file into a structure of the format DED_MAP_ENTRY.

The following are included:
"stdio.h"
"signal.h"
"ctype.h"
"sim_types.h"
"sim_dfns.h"
sim_macros.h"
"simstdio.h"
"libmap_dfn.h"
"libmap.h"

The following are declared as external:
map_start_names[NUMBER_OF_MAP_CLASSES] [30]
map_end_names[NUMBER_OF_MAP_CLASSES] [30]
ded_map_entries [MAX_DED_ENTRIES]
calloc()
skip_comment()
read_entry_attributes()
read_char()
read_long_int()
get_entries_until_end_subclass()
print_structure_contents()
check_for_defaults()
last_in_ammo_class [NUMBER_OF_MAP_CLASSES]
number_of_entries
map_file

2.6.11.3.1 map_file_read

This routine reads the file specified by the argument *file_name*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
file_name	pointer to char	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
temp_str [80]	char	Standard
Calls		
Function	**Where Described**	
skip_comment	Section 2.6.11.3.3	
get_entries_until_end_subclass	Section 2.6.11.3.6	
check_for_defaults	Section 2.6.11.3.7	
print_structure_contents	Section 2.6.11.3.8	

Table 2.6-179: map_file_read Information.

2.6.11.3.2 read_entry_attributes

This routine reads the attributes of the entries in the table.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp_str [80]	char	Standard
temp	int	Standard
Calls		
Function	**Where Described**	
read_char	Section 2.6.11.3.4	
read_long_int	Section 2.6.11.3.5	

Table 2.6-180: read_entry_attributes Information.

2.6.11.3.3 skip_comment

If a # is found at the beginning of a line, the line is skipped by searching for a carriage return.

Internal Variables		
Internal Variable	Type	Where Typedef Declared
character	int	Standard

Table 2.6-181: skip_comment Information.

2.6.11.3.4 read_char

This routine reads in a character string and compares it to a known string. If they match, the routine reads in a string and puts it in the structure.

Parameters		
Parameter	Type	Where Typedef Declared
compare_string	pointer to char	Standard
current_entry	int	Standard
data_entry	unsigned pointer to char	Standard
Internal Variables		
Internal Variable	Type	Where Typedef Declared
temp_str [30]	char	Standard
temp_int	int	Standard

Table 2.6-182: read_char Information.

2.6.11.3.5 read_long_int

This routine reads in a string and compares it to a known string. If they match, the routine reads in a long integer and puts it in the structure.

Parameters		
Parameter	Type	Where Typedef Declared
compare_string	pointer to char	Standard
current_entry	int	Standard
data_entry	pointer to int	Standard
Internal Variables		
Internal Variable	Type	Where Typedef Declared
temp_str [30]	char	Standard

Table 2.6-183: read_long_int Information.

2.6.11.3.6 get_entries_until_end_subclass

This routine reads in the entries of one particular class at a time. The subclass is designated by *subclass_num*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
subclass_num	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp_str [30]	char	Standard
Calls		
Function	**Where Described**	
skip_comment	Section 2.6.11.3.3	
read_entry_attributes	Section 2.6.11.3.2	

Table 2.6-184: get_entries_until_end_subclass Information.

2.6.11.3.7 check_for_defaults

This function checks for defaults.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard

Table 2.6-185: check_for_defaults Information.

2.6.11.3.8 print_structure_contents

This function is stubbed out but provides no functionality.

2.6.11.4 map_asid.c
(./simnet/release/src/libsrc/libmap/map_asid.c)

This file contains routines which are used to map between bits set in the appearance field of the VehicleAppearanceVariant to the cig ASID designators

The following are included:
"stdio.h"
"ctype.h"
"pro_sim.h"
"veh_type.h"
"veh_appear.h"
simstdio.h"

The following are declared as external:
dust_cloud_shift
dust_cloud_mask
dust_cloud_none
dust_cloud_small
dust_cloud_medium
dust_cloud_large
smoke_shift
flames_shift
tow_launcher_down_shift
tow_launcher_up_shift
engine_smoke_shift
bumper_mask
bumper_shift[3]
asid
bumper_status
asid_debug

2.6.11.4.1 map_read_asid_file

This routine reads an ASID mapping file designated by the argument *fn*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fn	pointer to char	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
fp	pointer to FILE	Standard
s[80]	char	Standard

Table 2.6-186: map_read_asid_file Information.

2.6.11.4.2 map_set_asid

This routine sets specific bits and values for the ASID

Parameters		
Parameter	**Type**	**Where Typedef Declared**
value	unsigned long	Standard
clear_mask	unsigned long	Standard

Table 2.6-187: map_set_asid Information.

2.6.11.4.3 map_clear_asid

This routine clears a specific bit for the ASID.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
value	unsigned long	Standard

Table 2.6-188: map_clear_asid Information.

2.6.11.4.4 map_set_bumper_numbers

This routine sets bits for vehicle bumper numbers. *marking* represents the bumper number to be set.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
marking	pointer to VehicleMarking	
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
ch[2]	char	Standard
Calls		
Function	**Where Described**	
map_set_asid	Section 2.6.11.4.2	

Table 2.6-189: map_set_bumper_numbers Information.

2.6.11.4.5 map_set_dust_cloud

This routine sets the dust cloud bits.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
cloud	int	Standard
Calls		
Function	**Where Described**	
map_set_asid	Section 2.6.11.4.2	

Table 2.6-190: map_set_dust_cloud Information.

2.6.11.4.6 map_get_bumper_status

This routine indicates whether or not bumper numbers are to be displayed.

Return Values		
Return Value	**Type**	**Meaning**
bumper_status	static int	whether or not bumper numbers are to be displayed

Table 2.6-191: map_get_bumper_status Information.

2.6.11.4.7 map_format_asid

This routine maps the ASID. This is the main routine in this module.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to VehicleAppearanceVariant	
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
appearance	register unsigned long	Standard
veh_type	ObjectType	p_sim.h
Return Values		
Return Value	**Type**	**Meaning**
asid	unsigned long	a properly formatted bit field(ASID) which tells the cig which effect to paint
Calls		
Function	**Where Described**	
veh_get_force	Section 2.6.10.6.3	
map_set_asid	Section 2.6.11.4.2	
map_clear_asid	Section 2.6.11.4.3	
map_set_dust_cloud	Section 2.6.11.4.5	
map_get_bumper_status	Section 2.6.11.4.6	
ia_friendly	Section 2.6.10.6.1	
map_set_bumper_numbers	Section 2.6.11.4.4	

Table 2.6-192: map_format_asid Information.

2.6.11.4.8 map_set_bumper_status

This routine sets the bits for bumper status.

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

Table 2.6-193 map_set_bumper_status Information.

2.6.11.5 map_veh.c
(./simnet/release/src/libsrc/libmap/map_veh.c)

This file contains the routines that read a vehicle map file into a structure of the format DED_VEH_ENTRY. This file was modified from read_map.c for ammo entries. The vehicle type is broken down into environments which are in turn broken down into classes.

The following files are included:

"stdio.h"
"signal.h"
"ctype.h"
"simstdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"obj_types.h"
"veh_appear.h"
"basic.h"
"libmap_dfn.h"
"libmap.h"

The structure DED_VEH_ENTRY is defined.

The following constants are defined:

MAX_VEH_MAP_CLASSES
MAX_NUM_DED_INDICIES
NUM_DEFAULTS
MAP_AIR_FIXED_WING
MAP_AIR_LIGHTER_THAN_AIR
MAP_AIR_ROTARY_WING
MAP_GROUND_SP_ARMORED_TRACKED
MAP_GROUND_SP_ARMORED_WHEELED
MAP_GROUND_SP_UNARMORED_TRACKED
MAP_GROUND_SP_UNARMORED_WHEELED
MAP_GROUND_TOWED
MAP_SPACE
MAP_WATER_AMPHIB_WARFARE
MAP_WATER_AUXILIARY
MAP_WATER_MATERIAL_SUPPORT
MAP_WATER_MINE_WARFVARE
MAP_WATER_SUBMARINE
JAP_WATER_SURFACE_COMBAT
JMAP_MUNITION
MAP_STRUCTURE
MAP_LIFEFORM

The following are declared as external:

veh_map_debug
veh_start_names[MAX_VEH_MAP_CLASSES] [40]
veh_end_names[MAX_VEH_MAP_CLASSES] [40]
start_veh_pt [MAX_VEH_MAP_CLASSES]
end_veh_pt [MAX_VEH_MAP_CLASSES]
ded_veh_entries [MAX_NUM_VEH_ENTRIES]
ded_cig_veh_ptrs [MAX_NUM_DED_INDICES]
calloc()
skip_veh_comment()
read_vehicle_entry_attributes()
read_char_vehicle_entry()
read_long_int_vehicle_entry()
get_vehicle_entries_until_end_subclass()
print_vehicle_attribute_contents()
check_for_vehicle_defaults()
entries_per_class [MAX_VEH_MAP_CLASSES]
number_of_entries
map_file

2.6.11.5.1 map_vehicle_file_read

This routine reads a vehicle mapping file designated by the argument *file_name*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
file_name	pointer to char	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
temp_str [80]	char	Standard
done	int	Standard
Calls		
Function	**Where Described**	
skip_veh_comment	Section 2.6.11.5.2	
get_vehicle_entries_until_end_subclass	Section 2.6.11.5.5	
check_for_vehicle_defaults	Section 2.6.11.5.26	

Table 2.6-194: map_vehicle_file_read Information.

2.6.11.5.2 read_vehicle_entry_attributes

This routine reads the attributes of the entries in the table. If the entry is ok, *cig_veh_type* is made to be an integer. *ded_cig_veh_ptrs[temp]* then points to the entry that was just added. This allows for rapid mapping of a *cig_veh_type* to an *object_type* for the network.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp_str[80]	char	Standard
temp	int	Standard
Calls		
Function	**Where Described**	
read_long_int_vehicle_entry	Section 2.6.11.5.5	
read_char_vehicle_entry	Section 2.6.11.5.4	

Table 2.6-195: read_vehicle_entry_attributes Information.

2.6.11.5.3 skip_veh_comment

If a # at the beginning of a line has been found, this line is ignored because it is a comment line.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
character	int	Standard

Table 2.6-196: skip_veh_comment Information.

2.6.11.5.4 read_char_vehicle_entry

This routine reads in a character string and compares it to a known string. If they match, the routine reads in a string and puts it in the structure.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
compare_string	pointer to char	Standard
curret_entry	int	Standard
data_entry	unsigned pointer to char	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp_str[30]	char	Standard
temp_int	int	Standard

Table 2.6-197: read_char_veh_entry Information.

2.6.11.5.5 read_long_int_vehicle_entry

This routine reads a string and compares it to a known string. If they match, the routine reads in a long integer and puts it in the structure.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
compare_string	pointer to char	Standard
current_entry	int	Standard
data_entry	pointer to int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp_str[30]	char	Standard

Table 2.6-198: read_long_int_veh_entry Information.

2.6.11.5.6 get_vehicle_entries_until_end_subclass

This routine reads in the entries of a particular vehicle class at a time. The vehicle subclass is designated by *subclass_num.*

Parameters		
Parameter	**Type**	**Where Typedef Declared**
subclass_num	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp_str[80]i	char	Standard
Calls		
Function	**Where Described**	
skip_veh_comment	Section 2.6.11.5.3	
read_vehicle_entry_attributes	Section 2.6.11.5.2	

Table 2.6-199 get_vehicle_entries_until_end_subclass Information.

2.6.11.5.7 check_for_vehicle_defaults

This routine checks for vehicle entry defaults. If the vehicle code doesn't exist, a default code will be assigned.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard

Table 2.6-200: check_for_vehicle_defaults Information.

2.6.11.5.8 map_net_to_cig

This routine takes the network type and returns the CIG type for various models. The network type is specified by *object_type* and *appearance*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
object_type	ObjectType	
appearance	unsigned long	Standard

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
check_for_match()	unsigned char	Standard

Return Values		
Return Value	**Type**	**Meaning**
check_for_match (MAP_MUNITION, object_type, appearance)	int	CIG model for a given munition
check_for_match (MAP_STRUCTURE, object_type, appearance)	int	CIG model for a building
check_for_match (MAP_LIFEFORM, object_type, appearance)	int	CIG model for a lifeform
check_for_match (MAP_AIR_FIXED_WING, object_type, appearance)	int	CIG model for a fixed wing aircraft
check_for_match (MAP_AIR_LIGHTER_THAN_AIR, object_type, appearance)	int	CIG model for any aircraft which is not a fused wing or rotary wing
check_for_match(MAP_AIR_ROTARY_WING, object_type, appearance)	int	CIG model for a rotary wing aircraft
check_for_match (MAP_GROUND_SP_ARMORED_TRACKED, object_type, appearance)	int	CIG model for an armored tracked ground vehicle
check_for_match (MAP_GROUND_SP_ARMORED_WHEELED, object_type, appearance)	int	CIG model for an armored wheeled ground vehicle
check_for_match (MAP_GROUND_SP_UNARMORED_TRACKED, object_type, appearance)	int	CIG model for an unarmored tracked ground vehicle
check_for_match (MAP_GROUND_SP_UNARMORED_WHEELED, object_type, appearance)	int	CIG model for an unarmored wheeled ground vehicle
check_for_match (MAP_GROUND_TOWED, object_type, appearance)	int	CIG model for a towed ground vehicle

Return Values (cont.)		
Return Value	**Type**	**Meaning**
check_for_match (MAP_WATER_AMPHIB_WARFARE, object_type, appearance)	int	CIG model for amphibious vehicle
check_for_match (MAP_WATER_AUXILIARY, object_type, appearance)	int	CIG model for a sea auxiliary vessel
check_for_match (MAP_WATER_MATERIAL_SUPPORT, object_type, appearance)	int	CIG model for sea supply vessel
check_for_match (MAP_WATER_MINE_WARFARE, object_type, appearance)	int	CIG model for a sea mine sweeper
check_for_match (MAP_WATER_SUBMARINE, object_type, appearance)	tint	CIG model for a submarine
check_for_match (MAP_WATER_SURFACE_COMBAT, object_type, appearance)	int	CIG model for a surface sea vessel
check_for_match (MAP_SPACE, object_type, appearance)	int	CIG model for a space craft

Calls	
Function	**Where Described**
check_for_match	Section 2.6.11.5.9

Table 2.6-201: map_net_to_cig Information.

2.6.11.5.9 check_for_match

This routine checks to see if the network type exists in the structure. If it doesn't, a default is chosen.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
subclass	int	Standard
object_type	ObjectTye	p_sim.h
appearance	unsigned long	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
tmpP	pointer to a pointer to DED_VEH_ENTRY	libmap_dfn.h
Return Values		
Return Value	**Type**	**Meaning**
(*tmpP)->destroyed_type	unsigned char	destroyed model
(*tmpP)->cig_veh_type	unsigned char	healthy model

Table 2.6-202: check_for_match Information.

2.6.12 libmem
(./simnet/release/src/libsrc/libmem [libmem])

Libmem provides a machine independent mechanism for establishing a logical shared memory segment in which control values are stored by the IDC device driver and from which they can be received by the simulation host.

2.6.12.1 assign_mp.c
(./simnet/release/src/libsrc/libmem/assign_mp.c)

This file contains routines which are called during initialization to set up the shared memory segment.

The following are included:

"stdio.h"
"fcntl.h"
"sys/types.h"
"sys/ipc.h"
"sys/pte.h"
"sys/shm.h"
"sim_dfns.h"
"fifo_dfn.h"
"libmem_dfn.h"
"libmem.h"

The following are defined:

IDCSHM_NAME (Butterfly only)

The following external variables are declared:

idc_values
cp

idc_values is a pointer to mapped memory.

2.6.12.1.1 mem_assign_memory_ptr

This routine is called at initialization to assign a pointer to the start address of the idc values.

If a Butterfly is used:

Calls	
Function	**Where Described**
map_idc_values	Section 2.6.12.1 .3

Table 2.6-203: mem_assign_memory_ptr Information for the Butterfly.

If a Masscomp is used:

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
id	int	Standard
fd	int	Standard
sbrk()	pointer to char	Standard
shmat()	pointer to char	Standard
size	int	Standard

Table 2.6-204: mem_assigned_memory_ptr Information.

2.6.12.1.2 mem_free_shared_memory

This routine frees the shared memory segment.

If a Butterfly is used:

Calls	
Function	**Where Described**
unmap_idc_values	Section 2.6.12.1.4

Table 2.6-205: mem_free_shared_memory Information for the Butterfly.

If a Masscomp is used no functions are called.

2.6.12.1.3 map_idc_values

This routine maps the idc values to shared memory. This routine is only defined for the Butterfly.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
idc_values_OID	OID	****

Table 2.6-206: map_idc_values Information.

2.6.12.1.4 unmap_idc_values

This routine unmaps the idc values from shared memory. This routine is only defined for the Butterfly.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
idc_values_OID	OID	****

Table 2.6-207: unmap_idc_values Information.

2.6.12.1.5 mem_get_idc_share_size

This routine returns the size of the IDC shared memory.

Return Values		
Return Value	**Type**	**Meaning**
IDC_SHARE_SIZE	int	IDC shared memory size

Table 2.6-208: mem_get_idc_share_size Information.

2.6.12.1.6 mem_get_memory_key

This routine returns the memory key.

Return Values		
Return Value	**Type**	**Meaning**
MEMORY_KEY	int	memory key

Table 2.6-209: mem_get_memory_key Information.

2.6.12.1.7 mem_get_total_share_size

This routine returns the size of the entire block of shared memory, which consists of the IDC segment and the FIFO segment.

Return Values		
Return Value	**Type**	**Meaning**
TOTAL_SHARE_SIZE	int	total shared memory size

Table 2.6-210: mem_get_total_share_size Information.

2.6.12.2 assign sm.c
(./simnet/release/src/libsrc/libmem/addign_sm.c)

This file contains a routine which assigns the shared memory segment at the startup of the simulation.

"libmem.h" is included.

2.6.12.2.1 mem_assign_shared_memory

This routine assigns the shared memory pointers used with the IDC's.

Calls	
Function	**Where Described**
mem_assign_memory_ptr	Section 2.6.12.1.1
mem_assign_other _ptrs	Section 2.6.13.1 for the M1 Section 2.6.14.1 for the M2 Section 2.6.15.1 for the Stealth

Table 2.6-211: mem_assign_shared_memory Information.

2.6.13 m1_mem.c
(/simnet/release/src/vehicle/m1/src/m1_mem.c [m1_mem.c])

Vehicle-specific routines are called in m1_mem.c to assign portions of the shared memory segment to individual idc ports, once the shared memory segment has been established.

Includes:
"stdio.h"
"sim_dfns.h"
"fifo_dfn.h"
"libmem_dfn.h"
"m1_mem_dfn.h"

Declared as pointers to FIFO:
fifo_driver
fifo_turret
fifo_ammo
dummy1
dummy2
sounds
dummy3
dummy4

Declared as pointer to short:
pser_heartbeat

fifo_driver is a pointer to the driver output queue. *fifo_turret* is a pointer to the turret output queue. *fifo_ammo* is a pointer to the ammo output queue. *dummy1* is a pointer to the dummy1 output queue. *dummy2* is a pointer to the dummy2 output queue. *sounds* is a pointer to the sounds output queue. *dummy3* is a pointer to the dummy3 output queue. *dummy4* is a pointer to the dummy4 output queue. *pser_heartbeat* is a pointer to the HPSM heartbeat.

2.6.13.1 mem_assign_other_ptrs

This routine assigns additional pointers to the shared memory segment for use in the M1 simulation.

2.6.14 m2_mem.c
(/simnet/release/ /vehicle/m2/src/m2_mem.c [m2_mem.c])

Vehicle-specific routines are called in m2_mem.c to assign portions of the shared memory segment to individual idc ports, once the shared memory segment has been established.

Includes:
- "stdio.h"
- "sim_dfns.h"
- "fifo_dfn.h"
- "libmem_dfn.h"
- "m2_mem_dfn.h"

Declared as pointers to FIFO:
- fifo_driver
- fifo_turret
- dummy1
- dummy2
- sounds
- dummy3
- dummy4
- alpha

Declared as pointer to short:
- pser_heartbeat

fifo_driver is a pointer to the driver output queue. *fifo_turret* is a pointer to the turret output queue. *dummy1* is a pointer to the dummy1 output queue. *dummy2* is a pointer to the dummy2 output queue. *sounds* is a pointer to the sounds output queue. *dummy3* is a pointer to the dummy3 output queue. *dummy4* is a pointer to the dummy4 output queue. *alpha* is a pointer to the alpha output queue. *pser_heartbeat* is a pointer to the HPSM heartbeat.

2.6.14.1 mem_assign_other_ptrs

This routine assigns additional pointers to the shared memory segment for use in the M2 simulation.

2.6.15 kato_mem.c
(/simnet/release/src/vehicle/kato/src/m2_mem.c [kato_mem.c])

Vehicle-specific routines are called in kato_mem.c to assign portions of the shared memory segment to individual idc ports, once the shared memory segment has been established.

Includes:
"stdio.h"
"sim_dfns.h"
"fifo_dfn.h"
"libmem_dfn.h"
"kato_mem_dfn.h"

Declared as pointers to FIFO:
fifo_soft
fifo_hard
dummy0
dummy1
dummy2
sounds
dummy3
dummy4

Declared as pointer to short:
pser_heartbeat

fifo_soft is a pointer to the soft panel output queue. *fifo_hard* is a pointer to the hard panel output queue. *dummy0* is a pointer to the dummy0 output queue. *dummy1* is a pointer to the dummy1 output queue. *dummy2* is a pointer to the dummy2 output queue. *sounds* is a pointer to the sounds output queue. *dummy3* is a pointer to the dummy3 output queue. *dummy4* is a pointer to the dummy4 output queue. *pser_heartbeat* is a pointer to the HPSM heartbeat.

2.6.15.1 mem_assign_other_ptrs

This routine assigns additional pointers to the shared memory segment for use by the Stealth.

2.6.16 librtc

This CSU contains a number of routines which are used to time segments of code This library can be used as a debugging feature.

2.6.16.1 rtc_timing.c

(./simnet/release/src/libsrc/librtc/rtc_timing.c)

This file contains routines which initialize a timing event, stop a timing event, It also contains routines which can be used to determine the length of time spent in a segment of code. Furthermore, routines are included which can be used for printing timing information in specific formats.

Includes:

"net/network.h"
"bbd_loc.h"
"rtc.h"

Defines:

NUMBER_IN_AVERAGE
TICK_RATE

External Variables:

rtc_bit_start[NUM_RTC_BITS]
rtc_values [NUM_RTC_BITS] [NUMBER_IN_AVERAGE]
current_timer_index [NUM_RTC_BITS]

rtc_bit_start[NUM_RTC_BITS] is the value of the timing bit at the start of the timing interval.
rtc_values [NUM_RTC_BITS] is the list of timing intervals for each timing bit.

If the simulation is running on a Masscomp, the following routines are stubbed out.

rtc_start_time()
rtc_stop_time()
rtc_time_history()
rtc_print_time()
rtc_print_overrun()
rtc_print1()
rtc_overrun()
rtc_print_permanent()
rtc_simul_history()

2.6.16.1.1 rtc_read_clock

This routine returns the number of *TICK_RATE* ticks since the timer was started.

Return Values		
Return Value	**Type**	**Meaning**
rtc	int	ticks elapsed since timer started for Butterfly machine
net_current_time (AssocGetNetHandle())	int	ticks elapsed since timer started for Masscomp machine
0	int	unknown system

Calls	
Function	**Where Described**
net_current_time	Section 2.20.2.8.3 in MCC CSCI
AssocGetNetHandle	Section **** MCC CSCI SDD

Table 2.6-212: rtc_read_clock Information.

2.6.16.1.2 rtc_start_time

This routine is called to mark the beginning of a timing interval. *bitnum* is the number of the timing bit.

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

Calls	
Function	**Where Described**
rtc_read_clock	Section 2.6.16.1.1

Table 2.6-213: rtc_start_time Information.

2.6.16.1.3 rtc_stop_time

This routine is called to end a timing interval. The value of the expired time is saved. *bitnum* is the number of the timing bit.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
bitnum	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
current_clock	register long	Standard
elapsed_ticks	register long	Standard
Calls		
Function	**Where Described**	
rtc_read_clock	Section 2.6.16.1.1	

Table 2.6-214: rtc_stop_time Information.

2.6.16.1.4 rtc_time_history

This routine prints the actual time spent each tick in the selected code. It prints out all of the saved entries for the given bit number. *bitnum* is the number of the timing bit, and *temp_str* is a string that will be printed before the time history. It is used to identify what is being printed.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
temp_str	pointer to char	Standard
bitnum	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
rtc_temp_time	register float	Standard

Table 2.6-215: rtc_time_history Information.

2.6.16.1.5 rtc_print_time

This routine prints out the maximum, minimum, and average amount of time spent in the selected segment of code. The average is taken over all saved entries for the given bit number.

bitnum is the number of the timing bit, and *temp_str* is a string that will be printed before the time history. It is used to identify what is being printed.

sum_of_times is the number of *TICK_RATE* ticks in the total timing interval. *min_ticks* is the number of *TICK_RATE* ticks in the smallest timing interval. *max_ticks* is the number of *TICK_RATE* ticks in the largest timing interval. *avg_time* is the average time spent in the subroutine in milliseconds. *min_time* is the smallest time spent in the subroutine in milliseconds. *max_time* is the largest amount of time spent in the segment of code in milliseconds.

The variables that compute the minimum, maximum, and average are initialized. The sum of the numbers to be averaged is found as well as the minimum and maximum timing inteervals. The average timing interval is then computed, ticks are converted to milliseconds, and the results are printed.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
bitnum	int	Standard
temp_str	pointer to char	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
sum_of_times	register long	Standard
min_ticks	register long	Standard
max_ticks	register long	Standard
avg_time	register float	Standard
min_time	register float	Standard
max_time	register float	Standard
i	register int	Standard

Table 2.6-216: rtc_print_time Information.

2.6.19.1.6 rtc_simul_history

A list of 10 segments of code have been declared always important. This routine prints all 50 saved measurements of the amount of time spent in the above 10 code segments.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
j	int	Standard
rtc_temp_time	float	Standard

Table 2.6-217: rtc_simul_history Information.

2.6.16.1.7 rtc_print_overrun

This routine prints the amount of time spent in any segment of code in which an overrun has occurred.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
over_run_ptr	int	Standard
i	int	Standard
Calls		
Function	**Where Described**	
rtc_overrun	Section 2.6.16.1.9	

Table 2.6-218: rtc_print_overrun Information.

2.6.16.1.8 rtc_printl

This routine is not used.

2.6.16.1.9 rtc_overrun

This routine finds the largest timing interval in the array of measurements and returns the index to this entry.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
longest_tick	int	Standard
longest_time	long	Standard
Return Values		
Return Value	**Type**	**Meaning**
longest_tick	int	pointer to longest timing interval

Table 2.6-219: rtc_overrun Information.

2.6.16.1.10 rtc_print_permanent

This routine prints out the amount of time spent in important segments of code.

Calls	
Function	**Where Described**
rtc_print_time	Section 2.6.16.1.5

Table 2.6-220: rtc_print_permanent Information.

2.6.16.1.11 rtc_get_tick_rate

This routine returns the tick rate for the real time clock on this machine.

Return Values		
Return Value	**Type**	**Meaning**
TICK_RATE	float	the tick rate for the rtc on this machine

Table 2.6-221: rtc_get_tick_rate Information.

2.6.16.1.12 rtc_get_start

This routine returns the start time of the specified timer. *bitnum* is the bit number of the specified timer.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
bitnum	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
rtc_bit_start[bitnum]	long	the start time of the specified timer

Table 2.6-222: rtc_get_start Information.

2.6.16.2 rtc.h

(./simnet/release/src/libsrc/librtc/rtc.h)

This file contains the defines for all timers used in the rtc functions.

2.6.17 libfile

(/simnet/release/src/libsrc/libfile [libfile])

This library provides a package of UNIX compatibility functions that don't exist on the Butterfly. Most of the functions are related to file access. These functions do exist on most UNIX systems. This code was written for partability.

2.6.18 libquat

Libquat performs equation of motion operations using the quaternions or Euler parameters method. Given accelerations and velocities, this library calculates the vehicle's dynamic equations.

2.6.18.1 calc_origin.c

Includes:
"sim_types.h"
"sim_dfns.h"

Declarations:
view_point

2.6.18.1.1 kinematics_viewpoint_offset

This routine sets the viewpoint offset to the value passed in *v*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v	VECTOR	/simnet/common/include/global/sim_types.h

Table 2.6-223: kinematics_viewpoint offset Information.

2.6.18.1.2 kinematics_calc_origin_state

This routine calculates the state variables of the origin. The model origin is not necessarily coincident with the center of gravity. First the angular velocity *w* is transformed into frame A. The velocity of the view point is calculated and converted to frame A. Parameters and variables are represented as follows:

B_w -- Angular velocity in frame B
B_v_cg -- Velocity of center of gravity in frame B
B_C_A -- Direction cosine matrix from frame B to frame A
A_w -- Angular velocity in frame A
A_v_o -- Velocity of the origin in frame A
B_v_o -- Velocity of the origin in frame B

Parameters		
Parameter	**Type**	**Where Typedef Declared**
B_w	VECTOR	/simnet/common/include/global/sim_types.h
B_v_cg	VECTOR	/simnet/common/include/global/sim_types.h
B_C_A	T_MATRIX	/simnet/common/include/global/sim_types.h
A_w	VECTOR	/simnet/common/include/global/sim_types.h
A_v_o	VECTOR	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
B_v_o	VECTOR	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
vec_mat_mul	Section 2.6.2.56.1
kinematics_calc_velocity	Section 2.6.18.2.1

Table 2.6-224: kinematics_calc_origin_state Information.

2.6.18.2 calc_v.c

Includes:
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"

2.6.18.2.1 kinematics_calc_velocity

This routine calculates the velocity at point b on Body B given the velocity at point a and the angular velocity of Body B using $v\_b = v\_a + w \times r$. Parameters and internal variables are represented as follows:

r -- Position vector to the point
v_a -- Velocity at point a
w -- Angular velocity of body B
v_b -- Velocity at point b
w_x_r -- Angular velocity (Omega) cross the position vector

Parameters		
Parameter	**Type**	**Where Typedef Declared**
r	VECTOR	/simnet/common/include/global/sim_types.h
v_a	VECTOR	/simnet/common/include/global/sim_types.h
w	VECTOR	/simnet/common/include/global/sim_types.h
v_b	VECTOR	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
w_x_r	VECTOR	/simnet/common/include/global/sim_types.h
Calls		
Function	**Where Described**	
vec_cross_prod	Section 2.6.2.66.1	
vec_add	Section 2.6.2.57.1	

Table 2.6-225: kinematics_calc_velocity Information.

2.6.18.3 form_c.x

"sim_types.h" and "sim_dfns.h" are included.

2.6.18.3.1 kinematics_form_C

This routine forms the Direction Cosine matrices from frame A to frame B and from frame B to frame A.

A_e_B -- Quaternion for rotation of Body B in frame A
A_c_B -- Direction Cosine matrix from frame A to frame B
B_c_A -- Direction Cosine matrix from frame B to frame A

Parameters		
Parameter	**Type**	**Where Typedef Declared**
A_e_B	pointer to REAL	/simnet/common/include/global/sim_types.h
A_c_B	T_MATRIX	/simnet/common/include/global/sim_types.h
B_c_A	T_MATRIX	/simnet/common/include/global/sim_types.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
temp	REAL	/simnet/common/include/global/sim_types.h
e0e0	REAL	/simnet/common/include/global/sim_types.h
e0e1	REAL	/simnet/common/include/global/sim_types.h
e0e2	REAL	/simnet/common/include/global/sim_types.h
e0e3	REAL	/simnet/common/include/global/sim_types.h
e1e1	REAL	/simnet/common/include/global/sim_types.h
e1e2	REAL	/simnet/common/include/global/sim_types.h
e1e3	REAL	/simnet/common/include/global/sim_types.h
e2e2	REAL	/simnet/common/include/global/sim_types.h
e2e3	REAL	/simnet/common/include/global/sim_types.h
e3e3	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_transpose	Section 2.6.2.51.1

Table 2.6-226: kinematics_form_C Information.

2.6.18.4 form_N.c

Includes:
"sim_types.h"
"sim_dfns.h"

2.6.18.4.1 kinematics_from_N

This routine forms the unit normal.

B_C_A -- Direction cosine matrix from B to A
A_b2 -- Normal vector in frame A

Parameters		
Parameter	**Type**	**Where Typedef Declared**
B_C_A	T_MATRIX	/simnet/common/include/global/sim_types.h
A_b2	VECTOR	/simnet/common/include/global/sim_types.h

Table 2.6-227: kinematics_form_N Information.

2.6.18.5 form_e.c

Includes:
"sim_types.h"
"sim_dfns.h"

2.6.18.5.1 kinematics_form_e

This routine forms the Euler parameters from w, the angular velocity. Note that all Euler parameters will be written in frame A.

A_w -- The angular speed vector
e -- The euler vector and parameter

Parameters		
Parameter	**Type**	**Where Typedef Declared**
A_w	VECTOR	/simnet/common/include/global/sim_types.h
e	pointer to REAL	/simnet/common/include/global/sim_types.h

Table 2.6-228: kinematics_form_e Information.

2.6.18.6 form_g.c

Includes:
"sim_types.h"
"sim_dfns.h"

2.6.18.6.1 kinematics_form_G

This routine forms the gravity vector in frame B.

A_c_B -- Direction cosine matrix from frame A to frame B
B_g -- Gravity vector in frame B

Parameters		
Parameter	**Type**	**Where Typedef Declared**
A_c_B	T_MATRIX	/simnet/common/include/global/sim_types.h
B_g	VECTOR	/simnet/common/include/global/sim_types.h

Table 2.6-229: kinematics_form_G Information.

2.6.18.7 form_r.c

Includes:
"sim_types.h"
"sim_dfns.h"

2.6.18.7.1 kinematics_form_r

This routine forms the position vector from B0* to B*.

A_v -- The velocity vector in frame A.
A_r -- A change in position vector, written in frame A.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
A_v	VECTOR	/simnet/common/include/global/sim_types.h
A_r	VECTOR	/simnet/common/include/global/sim_types.h

Table 2.6-230: kinematics_form_r Information.

2.6.18.8 form_s.c

Includes:
"sim_types.h"
"sim_dfns.h"
"libmatrix.h"

2.6.18.8.1 kinematics_form_s

This routine forms the position vector from B* to origin, where $B\_p = A\_p\, A\_c\_B$ and $B\_s = -B\_p$. Parameters and variables are represented as follows:

A_c_B -- Direction cosine matrix from frame A to frame B
A_p -- Position vector, from origin to B* in frame A
B_s -- Position vector, from B* to the origin in frame B
B_p -- Position vector in frame B

Parameters		
Parameter	**Type**	**Where Typedef Declared**
A_c_B	T_MATRIX	/simnet/common/include/global/sim_types.h
A_P	VECTOR	/simnet/common/include/global/sim_types.h
B_s	VECTOR	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
B_p	VECTOR	/simnet/common/include/global/sim_types.h
Calls		
Function	**Where Described**	
vec_mat_mul	Section 2.6.2.56.1	
vec_neg	Section 2.6.2.62.1	

Table 2.6-231: kinematics_form_s Information.

2.6.18.9 make_e.c

Includes:
"sim_types.h"
"sim_dfns.h"
"math.h"
"libmatrix.h"

Defines:
REAL_SMALL

Declarations:
C
e
denominator
line_no

2.6.18.9.1 make_e

This routine makes the Euler parameters from the Direction cosine matrix. Parameters are represented as follows:

C_in -- Direction cosine matrix
e_in -- Euler parameters

Parameters		
Parameter	**Type**	**Where Typedef Declared**
C_in	T_MATRIX	/simnet/common/include/global/sim_types.h
e_in	pointer to REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
mat_copy	Section 2.6.2.39.1
elr_copy	Section 2.6.2.3.1

Table 2.6-232: make_e Information.

2.6.18.9.2 quat_dump

This routine prints information about the quaternion equations for debugging purposes.

2.6.18.10 norm_e.c

Includes:
"math.h"
"sim_types.h"
"sim_dfns.h"

2.6.18.10.1 normalize_e

This routine normalizes the euler parameters, where *e[4]* represents the euler parameters.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
e[4]	REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
mag	REAL	/simnet/common/include/global/sim_types.h

Table 2.6-233: normalize_e Information.

2.6.18.11 update_e.c

Includes:
"sim_types.h"
"sim_dfns.h"

2.6.18.11.1 kinematics_update_e

This routine updates the Euler parameters. Parameters are represented as follows:

B0_e_B -- Euler parameters for rotation from B0 to B.
A_e_B0 -- Euler parameters for rotation from A to B0.
A_e_B -- Euler parameters for rotation from A to B.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
B0_e_B	pointer to REAL	/simnet/common/include/global/sim_types.h
A_e_B0	pointer to REAL	/simnet/common/include/global/sim_types.h
A_e_B	pointer to REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
normalize_e	Section 2.6.18.10.1

Table 2.6-234: kinematics_update_e Information.

2.6.18.1.2 update_p.c

Includes:
"sim_types.h"
"sim_dfns.h"
"libmatrix.h"

2.6.18.1.2.1 kinematics_update_p

This routine updates the position vector from origin to B*, where:
A_p[k] = A_p[k-1] + A_r[k-1]. Parameters are represented as follows:

A_v -- The velocity vector in frame A.
A_r -- Change in position, written in frame A.
A_p -- Position vector, from origin to B* in frame A.

Parameters		
Parameter	Type	Where Typedef Declared
A_v	VECTOR	/simnet/common/include/global/sim_types.h
A_r	VECTOR	/simnet/common/include/global/sim_types.h
A_p	VECTOR	/simnet/common/include/global/sim_types.h

Calls	
Function	Where Described
vec_add	Section 2.6.2.57.1

Table 2.6-235: kinematics_update_p Information.