(1)

SOFTWARE DESIGN DOCUMENT
Vehicle Simulation CSCI (5)

Volume 2 of 4 Sections 2.2.3.2 - 2.5.3.27.1

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

SOFTWARE DESIGN DOCUMENT
Vehicle Simulation CSCI (5)

Volume 2 of 4 Sections 2.2.3.2 - 2.5.3.27.1

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

2.2.3.2 m1_laser.c
(./simnet/release/src/vehicle/m1/src/m1_laser.c [m1_laser.c])

Includes:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"pro_data.h"
"libproc.h"
"libhull.h"
"libkin.h"
"libturret.h"
"libmsg.h"
"libfail.h"
"failure.h"
"m1_laser.h"
"m1_bcs.h"
"m1_cntrl.h"
"m1_elecsys.h"
"m1_firectl.h"

Defines:

Symbol	Value
PROBABILITY_OF_MULTIPLE_RETURNS	0.1
LRF_OFF	1
LRF_SAFE	2
LRF_ARMED_FIRST_RETURN	3
LRF_ARMED_LAST_RETURN	4

int declarations and initialization:
lrf_debug = 0
use_dazzler_laser = FALSE
lrf_state
number_of_lasings
lase_times[3]

BOOLEAN declarations:
its_dead_Jim
multiple_returns

REAL declarations:
value_from_cig

2.2.3.2.1 laser_init

This routine initializes the laser system.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
Calls		
Function	**Where Described**	
firectl_laser_malfunction_rese t	Section 2.2.2.2.3.4	
fail_init_failure	Section 2.5.4.11.2	

Table 2.2-73: laser_init Information.

2.2.3.2.2 laser_show_status

This routine prints status information for the laser system.

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

Table 2.2-74: laser_show_status Information.

2.2.3.2.3 laser_lrf_failure

This routine causes the laser range finder to fail.

Calls	
Function	**Where Described**
firectl_laser_malfunction_set	Section 2.2.2.2.3.3

Table 2.2-75: laser_lrf_failure Information.

2.2.3.2.4 laser_repair_lrf

This routine repairs the laser range finder.

Calls	
Function	**Where Described**
laser_init	Section 2.2.3.2.1
laser_show_status	Section 2.2.3.2.2

Table 2.2-76: laser_repair_lrf Information.

2.2.3.2.5 laser_power_off

This routine shuts the laser power off.

Errors	
Error Name	**Reason for Error**
laser_power_off(): unknown state	The variable lrf_state has an unexpected value.
Calls	
Function	**Where Described**
laser_show_status	Section 2.2.3.2.2

Table 2.2-77: laser_power_off Information.

2.2.3.2.6 laser_select_safe

This routine puts the laser into the LRF_SAFE state.

Errors	
Error Name	**Reason for Error**
laser_select_safe():unknown state	The variable lrf_state has an unexpected value.
Calls	
Function	**Where Described**
laser_show_status	Section 2.2.3.2.2
firectl_laser_malfunction_rese t	Section 2.2.2.2.3.4

Table 2.2-78: laser_select_safe Information.

2.2.3.2.7 laser_select_first_return

This routine puts the laser into the LRF_ARMED_FIRST_RETURN state.

Errors	
Error Name	**Reason for Error**
laser_select_first_return(): unknown state	The variable lrf_state has an unexpected value.
Calls	
Function	**Where Described**
laser_show_status	Section 2.2.3.2.2

Table 2.2-79: laser_select_first_return Information.

2.2.3.2.8 laser_select_last_return

This routine puts the laser into the LRF_ARMED_LAST_RETURN state.

Errors	
Error Name	**Reason for Error**
laser_select_last_return(): unknown state	The variable lrf_state has an unexpected value.
Calls	
Function	**Where Described**
laser_show_status	Section 2.2.3.2.2

Table 2.2-80: laser_select_last_return Information.

2.2.3.2.9 laser_power_up_safe

This routine powers up the laser and puts it into the LRF_SAFE state if it is off. Otherwise, a status message is printed.

Errors	
Error Name	**Reason for Error**
laser_power_up_safe(): unknown state	The variable lrf_state has an unexpected value.
Calls	
Function	**Where Described**
laser_show_status	Section 2.2.3.2.2

Table 2.2-81: laser_power_up_safe Information.

2.2.3.2.10 laser_power_up_first_return

This routine powers up the laser and puts it into the LRF_SAFE state if it is off. Otherwise, a status message is printed.

Errors	
Error Name	**Reason for Error**
laser_power_up_first_return() :unknown state	The variable lrf_state has an unexpected value.
Calls	
Function	**Where Described**
laser_init	Section 2.2.3.2.1
firectl_laser_malfunction_set	Section 2.2.2.2.3.3
laser_show_status	Section 2.2.3.2.2

Table 2.2-82: laser_power_up_first_return Information.

2.2.3.2.11 laser_power_up_last_return

This routine powers up the laser and puts it into the LRF_SAFE state if it is off. Otherwise, a status message is printed.

Errors	
Error Name	**Reason for Error**
laser_power_up_last_return(): unknown state	The variable lrf_state has an unexpected value.
Calls	
Function	**Where Described**
laser_init	Section 2.2.3.2.1
firectl_laser_malfunction_set	Section 2.2.2.2.3.3
laser_show_status	Section 2.2.3.2.2

Table 2.2-83: laser_power_up_last_return Information.

2.2.3.2.12 time_n_lases_ago

This routine returns the time *n* lases ago.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
n	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
MAXINT	int	n > number_of_lasings
timers_get [illegible]_tick() - lase_time[illegible]]	int	n <= number_of_lasings
Calls		
Function	**Where Described**	
timers_get_current_tick	Section 2.6.3.1.1	

Table 2.2-84: time_n_lases_ago Information.

2.2.3.2.13 record_this_lase

This rouitne records the time of the current lase.

Calls	
Function	**Where Described**
timers_get_current_tick	Section 2.6.3.1.1

Table 2.2-85: record_this_lase Information.

2.2.3.2.14 laser_perform_lase

This routine will attempt a lase. It returns TRUE if it tried to lase for range, regardless of success or failure. It returns FALSE if it tried to dazzle.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp[3]	REAL	sim_types.h
lased_pt	VECTOR	sim_types.h
target_id	TargetDescriptor	
Calls		
Function	**Where Described**	
electsys_laser_start_request	Section 2.2.6.3.1.10	
time_n_lases_ago	Section 2.2.3.2.12	
network_send_laser_range	Section 2.1.1.3.1.29.1	
laser_show_status	Section 2.2.3.2.2	
record_this_lase	Section 2.2.3.2.13	
roll_dice	sim_macros.h	
cig_laser_range	Section 2.1.2.2.3.4.1	
vec_mat_mul	Section 2.6.2.56.1	
kinematics_get_h_to_w	Section 2.5.8.1.2	
vec_add	Section 2.6.2.57.1	
kinematics_get_o_to_h	Section 2.5.8.1.4	
bcs_range_is	Section 2.2.3.1.11	

Table 2.2-86: laser_perform_lase Information.

2.2.3.2.15 laser_lase

This routine is an external call to attempt a lase.

Errors	
Error Name	**Reason for Error**
PANIC: laser_lase: laser in bad state: ...	The variable lrf_state has an unexpected value.
Calls	
Function	**Where Described**
laser_show_status	Section 2.2.3.2.2
laser_perform_lase	Section 2.2.3.2.14

Table 2.2-87: laser_lase Information.

2.2.3.2.16 laser_multiple_returns

This routine returns *multiple_returns*.

Return Values		
Return Value	**Type**	**Meaning**
multiple_returns	BOOLEAN	Normal end.

Table 2.2-88: laser_multiple_returns Information.

2.2.3.2.17 laser_ready_to_fire

This routine returns FALSE if the laser is dead or the lrf is off and returns TRUE otherwise.

Return Values		
Return Value	**Type**	**Meaning**
FALSE	BOOLEAN	The laser is dead or the lrf is off.
TRUE	BOOLEAN	The laser is alive and the lrf is on.

Table 2.2-89: laser_ready_to_fire Information.

2.2.3.2.18 laser_last_return

This routine returns TRUE if *lrf_state* equals LRF_ARMED_LAST_RETURN and returns FALSE otherwise.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	lrf_state == LRF_ARMED_ LAST_RETURN
FALSE	BOOLEAN	lrf_state != LRF_ARMED_ LAST_RETURN

Table 2.2-90: laser_last_return Information.

2.2.3.2.19 laser_range

This routine returns the laser range from the CIG.

Return Values		
Return Value	**Type**	**Meaning**
value_from_cig	REAL	the laser range.

Table 2.2-91: laser_range Information.

2.2.3.2.20 laser_fire control_malfunction

This routine causes the laser fire control to fail.

Return Values		
Return Value	**Type**	**Meaning**
its_dead_Jim	BOOLEAN	The laser fire control has failed

Table 2.2-92: laser_fire_control_malfunction Information.

2.2.3.3 m1_weapons.c

(./simnet/release/src/vehicle/m1/src/m1_weapons.c [m1_weapons.c])

This is the gunnery simulation module. Code for ballistics simulation is contained in this file.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"basic.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"libevent.h"
"libmatrix.h"
"libimps.h"
"libsusp.h"
"libkin.h"
"libhull.h"
"libnetwork.h"
"libturret.h"
"libsound.h"
"libball.h"
"libmath.h"
"libmap.h"
"mun_type.h"
"m1_ammo.h"
"m1_ammo_df.h"
"m1_cntrl.h"
"m1_sound_dfn.h"
"m1_turret.h"
"m1_firectl.h"
"m1_bcs.h"
"m1_cig.h"
"m1_laser.h"
"m1_weapons.h"

The following are defined:

GUN_MAIN	
GUN_SAFE	
GUN_COAX	
MAX_DGI_RANGE	meters
GUN_LENGTH	meters
GUN_HEIGHT	meters
RAD_STD_DEV	0.3 mils
WEAPONS_EMPTY	
EFFECT_OFFSET	offect in meters for origin of effect

The following variables are declared:

pser_heartbeat
ammo_type_loaded, ammo_type_in_flight
main_gun_firing_status, round_id
fire_position, fire_delta_position
fire_gun_to_world
null_vehicleID

2.2.3.3.1 weapons_download_ballistics_tables

This routine downloads the ballistics tables for the HEAT and SABOT rounds.

Variables		
Internal Variable	**Type**	**Where Typedef Declared**
using_assymetric	int	Standard
Calls		
Function	**Where Described**	
ballistics_load_trajectory_file	Section 2.5.2.3.1	

Table 2.2-93: weapons_download_ballistics_tables Information.

2.2.3.3.2 weapons_init

This routine initializes the weapons system.. The main gun firing status is set to WORKING, and the ammo type loaded is set to WEAPONS_EMPTY.

InternalVariables		
Internal Variable	**Type**	**Where Typedef Declared**
time_temp	long	Standard
Calls		
Function	**Where Described**	
ballistics_load_trajectory_file	Section 2.5.2.3.1	

Table 2.2-94: weapons_init Information.

2.2.3.3.3 weapons_simul

This routine is not used in the Version 6.6 release.

2.2.3.3.4 weapons_disable_main_gun

This routine disables the main gun by setting the main gun firing status to BROKEN.

2.2.3.3.5 weapons_repair_main_gun

This routine repairs the main gun by setting the main gun firing status to WORKING.

2.2.3.3.6 weapons_fire_main_gun

This is an event-driven function, used to fire the main gun. A sound fired message is generated and sent to the CIG. The CIG uses the downloaded ballistic tables to paint the picture of the round as it flies out.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
scaled_normal	VECTOR	sim_types.h
gun_muzzle	VECTOR	sim_types.h
bias_vector	VECTOR	sim_types.h
gun_velocity	VECTOR	sim_types.h
f_gun-velocity[3]	float	Standard
intersection_check_time	REAL	sim_types.h

Calls	
Function	**Where Described**
map_get_ammo_entry_from_network_type	Section 2.6.11.2.1
ammo_type_loaded_quick	Section 2.2.5.1.25
firectl_ready_to_fire	Section 2.2.2.2.3
ammo_gun_fired	Section 2.2.5.1.31
controls_gun_fired	Section 2.2.2
vec_scale	Section 2.6.2.64.1
kinematics_get_u_norm	Section 2.5.8.1.5
vec_add	Section 2.6.2.57.1
kinematics_get_o_to_h	Section 2.5.8.1.4
kinematics_get_d_pos	Section 2.5.8.1.7
bivariant_normal_distribution	Section 2.6.1.1.1
turret_get_gun_to_world	Section 2.2.6.1.1.36
event_get_eventid	Section 2.6.9.1.2
map_get_tracer_from_ammo_entry	Section 2.6.11.2.9
ballistics_fire_a_round	Section 2.5.2.2.1
vec_init	Section 2.6.2.61.1
vec_mat_mul	Section 2.6.2.56.1
impacts_queue_effect	Section 2.5.15.1.3
network_send_shell_fire_pkt	Section 2.1.1.3.1.57.1
laser_range	Section 2.2.3.2
turret_get_turret_slew_rate	Section 2.2.6.1.1.5
bcs_get_ammo_type	Section 2.2.3.2
suspension_gun_fired	Section 2.5.6.1.1
turret_to_hull_cos	
turret_to_hull_sin	
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-95: weapons_fire_main_gun Information.

2.2.4 M1 Failures

Failure generation in SIMNET allows the introduction of simulated physical failures and degradations of a vehicle's capabilities. This involves the generation of a variety of different kinds of failures and the presentation of these failures to either the crew of a manned simulator, or the operator of a computer controlled vehicle. Failures are divided into categories that indicate the method used for failure generation. The three categories are combat damage, stochastic failure, and deterministic failure.

Combat Damage

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.

Stochastic Failures

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.

Deterministic Failures

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.

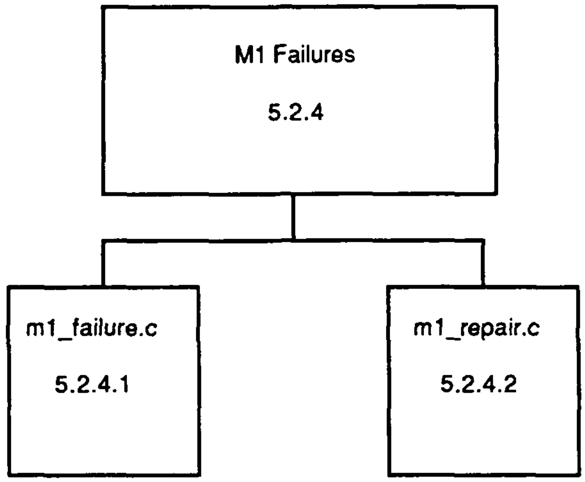

Figure 2.2-5: Structure of the M1 Failures CSC.

Simulation of these failure modes is accomplished with the following CSU's:

m1_failure.c
m1_repair.c

2.2.4.1 m1_failure.c

(./simnet/release/src/vehicle/m1/src/m1_failure.c [m1_failure.c])

The M1 specific failures functions are performed by this CSU.

m2_failure.c
m2_repair.c

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"pro_sim.h"
"mun_type.h"
"pro_data.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"libfail.h"
"failure.h"
"bigwheel.h"
"libsound.h"
"libmain.h"
"librva.h"
"libevent.h"
"m1_rep_map.h"
"m1_sound_dfn.h"
"libhull.h"
"librepair.h"
"libnetwork.h"
"repair_m1.h"
"status_m1.h"
"status.h"

Defines:

NUM_SUB_SYSTEMS
MOB_ELECTRICAL_SUBSYS
ENGINE_SUBSYS
TRANSMISSION_SUBSYS
MOB_OTHER_SUBSYS
FIRE_CONTROL_SUBSYS
GUN_TURRET_SUBSYS
ACTUAL_NUM_LEVELS
USED_NUM_LEVELS
MAINT_L_1_MMBF
MAINT_L_2_MMBF
MAINT_L_3_MMBF
MAINT_L_4_MMBF
MAINT_L_5_MMBF

Declarations:

MAINT_LEVEL_ARRAY

2.2.4.1.1 fail_init

This routine initializes the failures system. The combat failures and combat failure table are initialized, and the stochastic failures and stochastic damages table are initialized. The repairs are initialized and the collision damages are initialized.

Internal Variables		
Variable	Type	Where Typedef Declared
fialure_collision_damages	int	Standard

Calls	
Function	Where Described
failure_collision_damages	Section 2.2.4.1.8
cfail_init	Section 2.5.4.5.1
fail_table_init	Section 2.5.4.11.1
sfail_init	Section 2.5.4.23.1
get_sdamage_file	Section 2.5.1.2.7
lrepair_init	Section 2.5.14.5.1
collision_init	Section 2.5.10.5.1

Table 2.2-97: fail_init Information.

2.2.4.1.2 failure_mob_electrical_fixed

This routine is called by the repairs code when the alternator, pilot relay, or starter is fixed.

Calls	
Function	Where Described
sfail_fixed_good_as_new	Section 2.5.4.22.1

Table 2.2-98: failure_mob_electrical_fixed Information.

2.2.4.1.3 failure_engine_fixed

This routine is called by the repairs code when the engine oil filter, oil level, or major component is repaired or replaced.

Calls	
Function	Where Described
sfail_fixed_good_as_new	Section 2.5.4.22.1

Table 2.2-99: failure_engine_fixed Information.

2.2.4.1.4 failure_transmission_fixed

This routine is called by the repairs code when the transmission oil filter, oil level, or shifting assembly is repaired or replaced.

Calls	
Function	**Where Described**
sfail_fixed_good_as_new	Section 2.5.4.22.1

Table 2.2-100: failure_transmission_fixed Information.

2.2.4.1.5 failure_fuel_or_brakes_fixed

This routine is called by the repairs code when the fuel pump, fuel filter, service brake, or parking brake is repaired.

Calls	
Function	**Where Described**
sfail_fixed_good_as_new	Section 2.5.4.22.1

Table 2.2-101: failure_fuel_or_brakes_fixed Information.

2.2.4.1.6 failure_fire_control_fixed

This routine is called by the repairs code when the Laser Range Finder, Gunner's Primary Sight, or Gunner's Primary Sight Extension is fixed.

Calls	
Function	**Where Described**
sfail_fixed_good_as_new	Section 2.5.4.22.1

Table 2.2-102: failure_fire_control_fixed Information.

2.2.4.1.7 failure_gun_turret_fixed

This routine is called by the repairs code when the Gun/Turret Drive/Stab is fixed.

Calls	
Function	**Where Described**
sfail_fixed_good_as_new	Section 2.5.4.22.1

Table 2.2-103: failure_gun_turret_fixed Information.

2.2.4.1.8 failure_collision_damages

This routine is called by the collision code when a collision is detected. Parameters are represented as follows:

direction -- The direction the collision came from: RIGHT_WHEEL, LEFT_WHEEL, or REAR_WHEEL, as defined in /simnet/vehicle/libbigwh/bigwh_loc.h

cause -- The cause of the collision. *cause* can be either a hash id corresponding to the vehicle collided with or -2 (signifying a ground collision)

event_id -- The event id corresponding to the collision

Parameters		
Parameter	**Type**	**Where Typedef Declared**
direction	int	Standard
cause	int	Standard
event_id	long int	Standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
temp_failures	pointer to long	Standard
vid	pointer to VehicleID	basic.h

Return Values		
Return Value	**Type**	**Meaning**
0	int	damage detected

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2
rva_get_veh_id	Section 2.5.12.10.1
turret_collision_detected	Section 2.2.6.1.1.32
network_send_outta_my_way_mf	Section 2.1.1.3.1.8.1

Table 2.2-104: failure_collision_damages Information.

2.2.4.1.9 failure_check_cat_kill

This routine checks to see if a direct impact packet came from a mine. If the packet is from a mine, then the tank is destroyed. If the packet is not from a mine, the blast door is open, and the caliber of the round is greater than 50mm, then the tank is automatically destroyed. Otherwise, damages are calculated by calling **cfail_dir_fire_damages**() in libfail. Parameters are represented as follows:
ammo_type -- the ammo indix from libmap
hit_msg -- pointer to the Impact Variant PDU

Parameters		
Parameter	**Type**	**Where Typedef Declared**
hit_msg	pointer to ImpactVariant	p_sim.h
ammo_type	int	Standard

Calls	
Function	**Where Described**
ammo_blast_door_open	Section 2.2.5.1.43
fail_vehicle_is_destroyed	Section 2.5.4.9.2
cfail_dir_fire_damages	Section 2.5.4.3.1

Table 2.2-105: failure_check_cat_kill Information.

2.3.4.1.10 failure_check_indir_fire_damages

This routine checks to see if an indirect impact packet came from a mine. If the packet is from a mine and the detonation occurs within 10 meters of the tank, then the tank is destroyed. If the packet is not from a mine, then damages are calculated by calling **cfail_indirect_fire_damages**() in libfail. Parameters are represented as follows:
ammo_type -- the ammo indix from libmap
indir_fire_msg -- pointer to the indirect fire variant PDU
r_squared -- The range from the detonation to the vehicle
detonation_num -- The detonation number within the Indirect Fire Variant PDU

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_type	int	Standard
indir_fire_msg	pointer to IndirectFireVariant	p_sim.h
r_squared	REAL	sim_types.h
detonation_num	int	Standard

Calls	
Function	**Where Described**
fail_vehicle_is_destroyed	Section 2.5.4.9.2
cfail_indirect_fire_damages	Section 2.5.4.3.1
kinematics_get_h_to_o	Section 2.5.8.1.3
kinematics_get_w_to_h	Section 2.5.8.1.1

Table 2.2-106: failure_check_indir_fire_damages Information.

2.2.4.2 m1_repair.c
(./simnet/release/src/vehicle/m1/src/m1_repair.c [m1_repair.c])

The M1 specific repairs functions are performed by this CSU.

Includes:

"stdio.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"pro_assoc.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"pro_data.h"
"pro_sim.h"
"repair_m1.h"
"timers_dfn.h"
"timers.h"
"libnetwork.h"
"m1_cntrl.h"
"m1_resupp.h"

Defines:

QUIET
REQUEST
MAX_REPAIR_ENTITIES

Declarations:

repair_vehicle
repair_vehicles_near_hear
num_repair_vehicles
repair_state
repair_timer_id
clear_repair_vehicles()
repair_quiet_state()
repair_request_state()
send_feed_me_packets_repair_vehicles()

2.2.4.2.1 repair_request

This routine processes repair requests that come in over the network from the Admin/Log console of the MCC. The MCC host times these repairs and informs the simulator that the repair is complete via a network message.

Parameters		
Parameter	Type	Where Typedef Declared
event	int	Standard
agent	pointer to VehicleID	basic.h
code	int	Standard
originator	pointer to SimulationAddress	address.h
tid	TransactionIdentifier	p_assoc.h

Errors	
Error	Reason for Error
stderr, REPAIR: repair_request()	unknown repair state

Calls	
Function	Where Described
timers_free_timer	Section 2.6.3.5.1
repair_system_is_fixed	Section 2.5.4.19.4
send_repaired_pkt	Section 2.1.1.3.1.38.1

Table 2.2-107: repair_request Information.

2.2.4.2.2 repair_simul

This routine runs the repair simulation and is responsible for all self-repairs to the simulation The routine checks the vehicle's repair state and calls the appropriate routine for that state.

Errors	
Error	Reason for Error
stderr, REPAIR: repair_simul()	unknown repair state

Calls	
Function	Where Described
repair_quiet_state	Section 2.2.4.2.7
repair_request_state	Section 2.2.4.2.8
clear_repair_vehicles	Section 2.2.4.2.4

Table 2.2-108: repair_simul Information.

2.2.4.2.3 repair_init

This routine initializes the repair simulation. All repair vehicles are cleared, the repair state is set to QUIET, and the repair timer is set to NULL_TIMER.

Calls	
Function	**Where Described**
clear_repair_vehicles	Section 2.2.4.2.4

Table 2.2-109: repair_init Information.

2.2.4.2.4 clear_repair_vehicles

This routine clears the *repair_vehicles_near_here* flag to FALSE and the number of repair vehicles to zero.

2.2.4.2.5 repair_near_repair

This routine maintains the list of close vehicles which are repair vehicles. If any repair vehicles are on this list, the *repair_vehicles_near_here* flag is set to TRUE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v	pointer to VehicleID	basic.h

Table 2.2-110: repair_near_repair Information.

2.2.4.2.6 send_feed_me_packets_repair_vehicles

This routine sends a repair request packet to each of the repair vehicles on the network.

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

Calls	
Function	**Where Described**
network_send_feed_me_packet	2.1.1.3. [illegible].48.1

Table 2.2-111: send_feed_me_packets_repair_vehicles Information.

2.2.4.2.7 repair_quiet_state

This routine determines the vehicle's repair Finite State Machine's QUIET state. If the following conditions are BOTH TRUE:

- The resupply gating conditions are TRUE (the vehicle is alive, the vehicle is not moving, and no controls failures exist).
- There are repair vehicles nearby.

Then, send a feed me packet to the repair vehicles on the network, start the repair timer, and enter the REQUEST state. If any of the conditions are not met, remain in the QUIET state.

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	in QUIET state
REQUEST	int	in REQUEST state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.2.5.3.15
timers_get_timer	Section 2.6.3.6.1
send_feed_me_packets_repair_vehicles	Section 2.2.4.2.6

Table 2.2-112: repair_quiet_state Information.

2.2.4.2.8 repair_request_state

This routine determines the vehicle's repair Finite State Machine's REQUEST state. If EITHER of the following conditions are TRUE:

- The resupply gating conditions are FALSE (the vehicle is dead, the vehicle is moving, or a controls failure exists).
- There are no fuel carriers nearby.

Then, free the repair timer and enter the QUIET state. If none of the conditions are met and the resupply timer has expired, send another service request, restart the timer, and remain in the REQUEST state. If the resupply timer has not expired, remain in the REQUEST state.

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	in QUIET state
REQUEST	int	in REQUEST state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.2.5.3.15
timers_get_timer	Section 2.6.3.6.1
send_feed_me_packets_repair_vehicles	Section 2.2.4.2.6

Table 2.2-113: repair_request_state Information.

2.2.4.2.9 print_repair_status

This routine prints the repair status information.

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

Table 2.2-114: print_repair_status Information.

2.2.5 M1 Munitions Management

The simulation keeps track of fuel and ammunition in m1_ammo.c and m1_fuelsys.c. The fuel system can involve more than one fuel tank, and the desired fuel transfers within a vehicle. The simulation must also determine the states of all munitions on board a vehicle. This includes the availability, which ammunitions are loaded, and which ammunition is being fired. Resupply of all munitions (fuel and ammunition) is coordinated in the file m1_resupp.c. These files implement the functions necessary to resupply munitions from MCC vehicles (hemmits, etc.) and other vehicle simulations (m1 to m1). This functionality is realized by the following CSU's:

m1_ammo.c
m1_fuelsys.c
m1_resupp.c

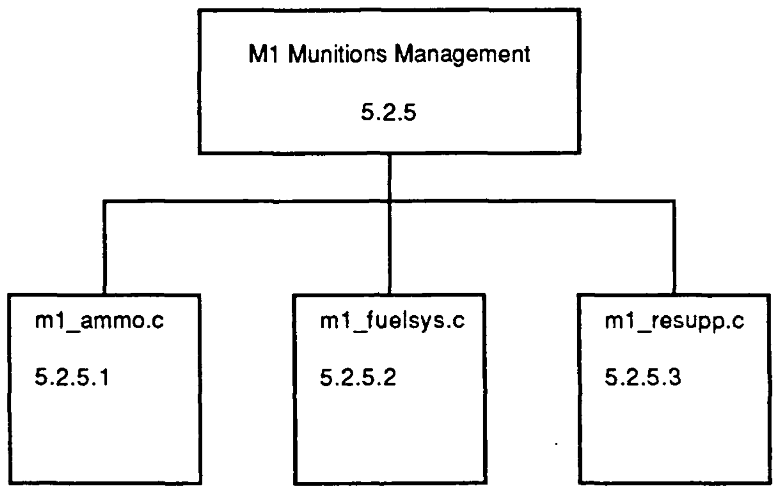

Figure 2.2-6: Structure of the M1 Munitions Management CSC.

2.2.5.1 m1_ammo.c

(./simnet/release/src/vehicle/m1/src/m1_ammo.c [m1_ammo.c])

This CSU keeps track of the M1's ammunition supply. This file contains the procedures relating to the supply, resupply, and loading of ammunition. The ammo subsystem of the M1 simulation consists of the loader's ammo rack, the semi-ready ammo rack, the hull ammo rack, the loader's knee switch, the loader's arms, the breech, the ejection guard, and the blast door.

Includes:

"stdio.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"timers_dfn.h"
"timers.h"
"libnetwork.h"
"libsound.h"
"m1_ammo.h"
"m1_ammo_df.h"
"m1_ammo_mx.h"
"m1_ammo_pn.h"
"m1_cntrl.h"
"m1_ctl_df.h"
"m1_firectl.h"
"m1_gunn_mx.h"
"m1_hydrsys.h"
"m1_load_mx.h"
"m1_resupp.h"
"m1_sound_dfn.h"
"m1_tmrs_df.h"
"m1_turr_pn.h"
"mun_type.h"

Variable and Local Procedure Declarations:

loaders_ready_rack -- The loader's ready rack is viewed as an array of values which are either munition_US_M456A1, munition_US_M392A2, or EMPTY

-- The next six variables keep track of the number of rounds of each type.

hull_apds_quantity
hull_heat_quantity
semi_ready_apds_quantity
semi_ready_heat_quantity
ready_apds_quantity
ready_heat_quantity

initial_heat_quantity
initial_apds_quantity

The next variables keep the status of various subsystems

ammo_transfer_status	
blast_door_status	-- OPEN or CLOSED
loaders_arms	-- munition_US_M456A1, munition_US_M392A2, or EMPTY
breech_ammo_type	-- munition_US_M456A1, munition_US_M392A2, or EMPTY
ejection_guard_status	-- SAFE or ARMED
autoloader_enabled	

turret_power_status
knee_switch_status
loader_timer_number
blast_door_timer_number
resupply_receive_timer_number
resupply_status
resupply_slot
resupply_flash_count
resupply_location
resupply_send_in_progress

ammo_open_blast_door()
ammo_close_blast_door()
ammo_arm_panel_check()
ammo_resupply_check()
ammo_add_round()
ammo_subtract_round()
ammo_start_loader_timer()
ammo_stop_loader_timer()
ammo_start_blast_door_timer()
ammo_stop_blast_door_timer()
ammo_start_resupply_receive_timer()
ammo_stop_resupply_receive_timer()
ammo_blast_door_check()
ammo_resupply_receive_timeout_check()
ammo_flash_check()
ammo_start_internal_resupply()
ammo_change_resupply()
ammo_decide_resupply_receive()
ammo_decide_receive_location()
ammo_decide_resupply_send()
ammo_decide_resupply_slot()
ammo_check_autoloader_unload()
ammo_check_autoloader_load()

2.2.5.1.1 ammo_init

This routine initializes the ammunition system states. Initial ammunition information is provided by the MCC during the activation sequence.

Calls	
Function	**Where Described**
controls_ejection_guard_safe	Section 2.2.2
timers_set_null_timer	Section 2.6.3.1.4.1

Table 2.2-115: ammo_init Information.

2.2.5.1.2 ammo_simul

This routine simulates the functions of the ammunition system on a tick by tick basis. The routine checks the status of the blast door and the status of any internal or external ammunition transfers or resupplies. The routine monitors which gun is selected, and coordinates the ammunition types with the ballistics computer system (BCS).

Calls	
Function	**Where Described**
ammo_blast_door_check	Section 2.2.5.1.51
ammo_flash_check	Section 2.2.5.1.52
ammo_resupply_receive_timeout_check	Section 2.2.5.1.53
bcs_get_ammo_type_indexed	Section 2.2.6.2
firectl_gun_select_status	Section 2.2.2.2.3
ammo_check_autoloader_unload	Section 2.2.5.1.3
ammo_check_autoloader_load	Section 2.2.5.1.4

Table 2.2-116: ammo_simul Information.

2.2.5.1.3 ammo_check_autoloader_unload

The autoloader is enabled with the command line -A in order to perform ammo transfers without a person acting as loader. This routine unloads the breech and replaces the round in the ready rack when the autoloader is enabled. The routine checks to see if the breech is full. If so, the ejection guard is opened, the blast door is opened, and the breech round is removed and replaced in a slot..

Calls	
Function	**Where Described**
ammo_ejection_guard_safe	Section 2.2.5.1.27
ammo_breech_unload_pushed	Section 2.2.5.1.24
ammo_knee_switch_on	Section 2.2.5.1.3
ammo_tube_selected	Section 2.2.5.1.11
ammo_decide_resupply_slot	Section 2.2.5.1.68
ammo_knee_switch_off	Section 2.2.5.1.10

Table 2.2-117: ammo_check_autoloader_unload Information.

2.2.5.1.4 ammo_check_autoloader_load

The autoloader is enabled with the command line -A in order to perform ammo transfers without a person acting as loader. This routine loads the breech when the autoloader is enabled. The routine checks to see if the breech is empty. If so, the ejection guard is opened, the blast door is opened, a round is selected from the ready rack and placed in the breech. Then, the blast door is closed, the breech load button is pushed, and the ejection guard is armed.

Calls	
Function	**Where Described**
ammo_ejection_guard_safe	Section 2.2.5.1.27
ammo_knee_switch_on	Section 2.2.5.1.3
bcs_get_ammo_type_indexed	Section 2.2.6.2
ammo_tube_selected	Section 2.2.5.1.11
ammo_knee_switch_off	Section 2.2.5.1.10
ammo_breech_pushed	Section 2.2.5.1.23
ammo_ejection_guard_armed	Section 2.2.5.1.26

Table 2.2-118: ammo_check_autoloader_load Information.

2.2.5.1.5 ammo_init_ammo_racks

This routine initializes the quantities and types of ammo in the ready rack, the semi-ready rack, and the hull rack. The routine checks to make the ammo racks are not overloaded, and for the ready rack, places the ammo into individual slots in the ready rack. Parameters are represented as follows:

ready_heat -- The quantity of heat rounds to place in the ready rack.
ready_apds -- The quantity of apds rounds to place in the ready rack.
semi_ready_heat -- The quantity of heat rounds to place in the semi-ready rack.
semi_ready_apds -- The quantity of apds rounds to place in the semi-ready rack.
hull_heat -- The quantity of heat rounds to place in the hull.
hull_apds -- The quantity of apds rounds to place in the hull.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ready_heat	int	Standard
ready_apds	int	Standard
semi_ready_heat	int	Standard
semi_ready_apds	int	Standard
hull_heat	int	Standard
hull_apds	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	register int	Standard

Table 2.2-119: ammo_init_ammo_racks Information.

2.2.5.1.6 ammo_supply_full

This routine returns TRUE if all racks are full and FALSE if they are not all full.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	The ammo supply is full
FALSE	int	The ammo supply is not full

Table 2.2-120: ammo_supply_full Information.

2.2.5.1.7 ammo_supply_empty

This routine returns TRUE if all racks are empty and FALSE if they are not all empty.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	The ammo supply is empty
FALSE	int	The ammo supply is not empty

Table 2.2-121: ammo_supply_empty Information.

2.2.5.1.8 ammo_loaders_arms

This routine returns the status of the *ammo_loaders_arms*, either filled with a heat round, an apds round, or EMPTY.

Return Values		
Return Value	**Type**	**Meaning**
loaders_arms	ObjectType	the position of the loaders_arms: either munition_US_M456A1. munition_US_393A2, or EMPTY

Table 2.2-122: ammo_supply_full Information.

2.2.5.1.9 ammo_knee_switch_on

This routine is called when the knee switch is hit. If the knee switch status is OFF, the status is changed to ON and the loader timer is stopped. Since the knee switch is only used for transfers to and from the ready rack; the transfer status must be NO_TRANSFER_VAL for anything to happen.

If the turret power is off or in the process of being shut off (*turret_power_status* = OFF or OFF_EDGE), hitting the knee switch will have no effect. If the turret power is on or in the process of turning on (*turret_power_status* = ON or ON_EDGE), the blast door is opened and the blast door timer is stopped.

Errors	
Error	**Reason for Error**
stderr AMMO: ammo_knee_switch_on	Impossible ammo transfer status
Calls	
Function	**Where Described**
ammo_stop_loader_timer	Section 2.2.5.1.48
hydraulic_ammo_door_open_request	Section 2.2.6.4.2.7
ammo_open_blast_door	Section 2.2.5.1.29
ammo_stop_blast_door_timer	Section 2.2.5.1.50
ammo_start_blast_door_timer	Section 2.2.5.1.49

Table 2.2-123: ammo_knee_switch_on Information.

2.2.5.1.10 ammo_knee_switch_off

This routine is called when the knee switch is released. If the knee switch status is ON, the status is changed to OFF and the loader timer is started. Since the knee switch is only used for transfers to and from the ready rack; the transfer status must be NO_TRANSFER_VAL for anything to happen.

If the turret power is off or in the process of being shut off (*turret_power_status* = OFF or OFF_EDGE), releasing the knee switch will have no effect. If the turret power is on or in the process of turning on (*turret_power_status* = ON or ON_EDGE), the blast door timer is started.

Errors	
Error	**Reason for Error**
stderr AMMO: ammo_knee_switch_on	Impossible ammo transfer status
Calls	
Function	**Where Described**
ammo_start_loader_timer	Section 2.2.5.1.47
ammo_start_blast_door_timer	Section 2.2.5.1.49

Table 2.2-124: ammo_knee_switch_off Information.

2.2.5.1.11 ammo_tube_selected

This routine is called when an ammo slot in the ready rack has been selected. The slot is specified by *slot*. If the blast door is closed, it is impossible to touch that ammo slot and the routine does nothing. If the blast door is open and the transfer status is NO_TRANSFER_VAL, the loader replaces the ammo into the ready rack; **ammo_arm_panel_check** is called. For any other transfer status, then loader replaces the ammo into the specified rack; **ammo_resupply_check** is called.

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

Calls	
Function	**Where Described**
ammo_arm_panel_check	Section 2.2.5.1.12
ammo_resupply_check	Section 2.2.5.1.13

Table 2.2-125: ammo_tube_selected Information.

2.2.5.1.12 ammo_arm_panel_check

This routine is checked each tick of the simulation in order to manage ammo in the ready rack.

If the specified ready rack *slot* is full and the loaders arms are full, the routine does nothing. The round contained in the loaders arms cannot be placed into the full slot, and the round from the slot cannot be placed in the loaders arms.

If the loaders arms are empty and the specified *slot* is full, the round is removed from the slot and placed in the loaders arms. The sound of the ready rack extract is made, the slot status is set to EMPTY, and the loaders arms status are set to FULL.

If the loaders arms are full and the specified *slot* is empty, the round taken from the loaders arms and placed in the slot. The correct sound of the ready rack insert is made, the loaders arms are set to EMPTY, and the slot status is set to FULL.

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

Calls	
Function	**Where Described**
controls_unshow_round	Section 2.2.2
ammo_subtract_round	Section 2.2.5.1.42
sound_make_const_sound	Section 2.1.3.1.2
controls_show_round	Section 2.2.2
ammo_add_round	Section 2.2.5.1.41

Table 2.2-126: ammo_arm_panel_check Information.

2.2.5.1.13 ammo_resupply_check

This routine is checked each tick of the simulation in order to manage the resupply of ammo. If the loaders arms are full and if the specified *slot* is full, the loader cannot resupply ammunition and the routine does nothing.

If the loaders arms are empty and the transfer status is equal to HULL_HEAT_VAL, HULL_APDS_VAL, SEMI_HEAT_VAL, or SEMI_APDS_VAL and the quantity of that type in that rack is greater than zero, then start the internal resupply from the ammo type and rack specified in the transfer status to the specified *slot*.

If the loaders arms are empty and the transfer status is REDIST_RECV_VAL, then receive the external resupply ammo type into the *slot*.

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

Calls	
Function	**Where Described**
ammo_change_resupply	Section 2.2.5.1.56
ammo_start_internal_resupply	Section 2.2.5.1.54

Table 2.2-127: ammo_resupply_check Information.

2.2.5.1.14 ammo_get_quantity

This routine returns the quantity of the ammo in the rack specified by the *type* identifier.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
type	char	Standard

Return Values		
Return Value	**Type**	**Meaning**
hull_heat_quanity	int	The number of heat rounds in the hull rack.
hull_apds_quanity	int	The number of apds rounds in the hull rack.
semi_ready_heat_quantity	int	The number of heat rounds in the semi-ready rack.
semi_ready_apds_quantity	int	The number of apds rounds in the semi-ready rack.
ready_heat_quantity	int	The number of heat rounds in the ready rack.
ready_apds_quantity	int	The number of apds rounds in the ready rack.
NULL	int	error; impossible transfer type

Errors	
Error	**Reason for Error**
stderr AMMO: ammo_get_quantity	impossible transfer type

Table 2.2-128: ammo_get_quantity Information.

2.2.5.1.15 ammo_transfer_semi_heat

This routine starts the transfer of a heat round from the semi-ready rack when the selector on the Ammo Redistribution Panel is placed in the Semi-Ready Heat position. The transfer status is changed to SEMI_HEAT_VAL, any other resupply in progress is stopped, and the blast door is opened.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.2.5.1.57
ammo_open_blast_door	Section 2.2.5.1.29
ammo_stop_blast_door_timer	Section 2.2.5.1.50

Table 2.2-129: ammo_transfer_semi_heat Information.

2.2.5.1.16 ammo_transfer_semi_apds

This routine starts the transfer of an apds round from the semi-ready rack when the selector on the Ammo Redistribution Panel is placed in the Semi-Ready Sabot position. The transfer status is changed to SEMI_APDS_VAL, any other resupply in progress is stopped, and the blast door is opened.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.2.5.1.57
ammo_open_blast_door	Section 2.2.5.1.29
ammo_stop_blast_door_timer	Section 2.2.5.1.50

Table 2.2-130: ammo_transfer_semi_apds Information.

2.2.5.1.17 ammo_transfer_hull_heat

This routine starts the transfer of a heat round from the hull rack when the selector on the Ammo Redistribution Panel is placed in the Hull/Turret Floor Heat position. The transfer status is changed to HULL_HEAT_VAL, any other resupply in progress is stopped, and the blast door is opened.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.2.5.1.57
ammo_open_blast_door	Section 2.2.5.1.29
ammo_stop_blast_door_timer	Section 2.2.5.1.50

Table 2.2-131: ammo_transfer_hull_heat Information.

2.2.5.1.18 ammo_transfer_hull_apds

This routine starts the transfer of an apds round from the hull rack when the selector on the Ammo Redistribution Panel is placed in the Hull/Turret Floor Sabot position. The transfer status is changed to HULL_APDS_VAL, any other resupply in progress is stopped, and the blast door is opened.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.2.5.1.57
ammo_open_blast_door	Section 2.2.5.1.29
ammo_stop_blast_door_timer	Section 2.2.5.1.50

Table 2.2-132: ammo_transfer_hull_apds Information.

2.2.5.1.19 ammo_transfer_no_transfer

This routine stops the transfer of ammo when the selector on the Ammo Redistribution Panel is placed in the Off position. The transfer status is changed to NO_TRANSFER_VAL, any resupply in progress is stopped, and the blast door timer is started in order to close the blast door.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.2.5.1.57
ammo_start_blast_door_timer	Section 2.2.5.1.49

Table 2.2-133: ammo_transfer_no_transfer Information.

2.2.5.1.20 ammo_transfer_redist_send

This routine initiates the sending of ammo to another M1 tank when the selector on the Ammo Redistribution Panel is placed in the Ammo Transfer Send position. The transfer status is changed to REDIST_SEND_VAL, any resupply in progress is stopped, and the blast door is opened.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.2.5.1.57
ammo_open_blast_door	Section 2.2.5.1.29
ammo_stop_blast_door_timer	Section 2.2.5.1.50

Table 2.2-134: ammo_transfer_redist_send Information.

2.2.5.1.21 ammo_transfer_redist_recv

This routine initiates the receiving of ammo from another vehicle when the selector on the Ammo Redistribution Panel is placed in the Ammo Transfer Rec position. The transfer status is changed to REDIST_RECV_VAL, any resupply in progress is stopped, and the blast door is opened.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.2.5.1.57
ammo_open_blast_door	Section 2.2.5.1.29
ammo_stop_blast_door_timer	Section 2.2.5.1.50

Table 2.2-135: ammo_transfer_redist_recv Information.

2.2.5.1.22 ammo_get_transfer_status

This routine returns the transfer status, which reflects the position of the selector switch on the Ammo Redistribution Panel.

Return Values		
Return Value	Type	Meaning
ammo_transfer_status	char	The transfer status.

Table 2.2-136: ammo_get_transfer_status Information.

2.2.5.1.23 ammo_breech_pushed

This routine handles the loading of the breech when the red Breech Load button is pushed. If the breech is ready, the round is transferred from the loaders arms to the breech. The status of the loaders arms is set to EMPTY, and the sound of the breech closing is made.

Calls	
Function	Where Described
ammo_breech_ready	Section 2.2.5.1.23
controls_show_breech	Section 2.2.2
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-137: ammo_breech_pushed Information.

2.2.5.1.24 ammo_breech_unload_pushed

This routine handles the unloading of the breech when the Breech Operating Toggle Switch is clicked down. The loaders arms must be empty, the breech must be loaded, the ejection guard must be open, and the system must not be in resupply in order to unload the breech. If any of these conditions are not met, this routine does nothing. If the conditions are met, the round is transferred from the breech to the loaders arms. The ammo type in the breech is set to EMPTY, and the sound of the breech opening is made.

Calls	
Function	Where Described
controls_show_breech	Section 2.2.2
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-138: ammo_breech_unload_pushed Information.

2.2.5.1.25 ammo_type_loaded_quick

This routine returns the ammo type loaded in the breech.

Return Values		
Return Value	Type	Meaning
breech_ammo_type	ObjectType	The type of ammo in the breech.

Table 2.2-139: ammo_type_loaded_quick Information.

2.2.5.1.26 ammo_ejection_guard_armed

This routine arms the ejection guard, making the closing sound.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-140: ammo_ejection_guard_armed Information.

2.2.5.1.27 ammo_ejection_guard_safe

This routine opens the ejection guard, making the opening sound.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-141: ammo_ejection_guard_safe Information.

2.2.5.1.28 ammo_ejection_guard_status

This routine returns the ejection guard status.

Return Values		
Return Value	**Type**	**Meaning**
ejection_guard_status	char	the status of the ejection guard: either SAFE or ARMED

Table 2.2-142: ammo_ejection_guard_status Information.

2.2.5.1.29 ammo_open_blast_door

This routine opens the blast door, making the appropriate sound. Each round in the ready rack is shown with its slot's light turned on, indicating the round type (either Heat or Sabot). Empty slots in the ready rack will have their lights turned off.

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

Calls	
Function	**Where Described**
controls_show_round	Section 2.2.2
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-143: ammo_open_blast_door Information.

2.2.5.1.30 ammo_close_blast_door

This routine opens the blast door, making the appropriate sound. All ready rack slot indicator lights turn off.

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

Calls	
Function	**Where Described**
controls_unshow_round	Section 2.2.2
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-144: ammo_close_blast_door Information.

2.2.5.1.31 ammo_gun_fired

This routine empties the breech when the gun is fired. The ammo type in the breech is set to EMPTY.

Calls	
Function	**Where Described**
controls_show_breech	Section 2.2.2

Table 2.2-145: ammo_gun_fired Information.

2.2.5.1.32 ammo_ready_to_fire

This routine returns whether the ejection guard is armed or safe. If the ejection guard is armed and the gun is ready to fire, the routine returns TRUE; if the ejection guard is safe and the gun is not ready to fire, the routine returns FALSE.

Return Values		
Return Value	**Type**	**Meaning**
ammo_ejection_guard_status() == ARMED	BOOLEAN	if TRUE, the ejection guard is armed and the gun is ready to fire; if FALSE, the ejection guard is safe and the gun is not ready to fire.

Calls	
Function	**Where Described**
ammo_ejection_guard_status	2.2.5.1.28

Table 2.2-146: ammo_ready_to_fire Information.

2.2.5.1.33 ammo_get_semi_heat_quantity

This routine gets the quantity of heat rounds in the semi-ready rack.

Return Values		
Return Value	**Type**	**Meaning**
semi_ready_heat_quantity	int	The quantity of heat rounds in the semi-ready rack

Table 2.2-147: ammo_get_semi_heat_quantity Information.

2.2.5.1.34 ammo_get_semi_apds_quantity

This routine returns the quantity of apds rounds in the semi-ready rack.

Return Values		
Return Value	**Type**	**Meaning**
semi_ready_apds_quantity	int	The quantity of apds rounds in the semi-ready rack.

Table 2.2-148: ammo_get_semi_apds_quantity Information.

2.2.5.1.35 ammo_get_hull_heat_quantity

This routine returns the quantity of heat rounds in the hull rack.

Return Values		
Return Value	**Type**	**Meaning**
hull_heat_quantity	int	The quantity of heat rounds in the hull rack.

Table 2.2-149: ammo_get_hull_heat_quantity Information.

2.2.5.1.36 ammo_get_hull_apds_quantity

This routine returns the quantity of apds rounds in the hull rack.

Return Values		
Return Value	**Type**	**Meaning**
hull_apds_quantity	int	The quantity of apds rounds in the hull rack.

Table 2.2-150: ammo_get_hull_apds_quantity Information.

2.2.5.1.37 ammo_get_ready_heat_quantity

This routine returns the quantity of heat rounds in the ready rack.

Return Values		
Return Value	**Type**	**Meaning**
ready_heat_quantity	int	The quantity of heat rounds in the ready rack.

Table 2.2-151: ammo_get_ready_heat_quantity Information.

2.2.5.1.38 ammo_get_ready_apds_quantity

This routine returns the quantity of apds rounds in the ready rack.

Return Values		
Return Value	**Type**	**Meaning**
ready_apds_quantity	int	The quantity of apds rounds in the ready rack.

Table 2.2-152: ammo_get_ready_apds_quantity Information.

2.2.5.1.39 ammo_get_heat105_quantity

This routine gets the quantity of heat rounds in the ready rack, the semi-ready rack, and the hull rack combined.

Return Values		
Return Value	**Type**	**Meaning**
ready_heat_quantity + semi_ready_heat_quantity + hull_heat_quantity	int	The total number of heat rounds.

Table 2.2-153: ammo_get_heat105_quantity Information.

2.2.5.1.40 ammo_get_apds105_quantity

This routine gets the quantity of apds rounds in the ready rack, the semi-ready rack, and the hull rack combined.

Return Values		
Return Value	**Type**	**Meaning**
ready_apds_quantity + semi_ready_apds_quantity + hull_apds_quantity	int	The total number of apds rounds.

Table 2.2-154: ammo_get_apds105_quantity Information.

2.2.5.1.41 ammo_add_round

This routine increments the ready rack quantity of the type of round passed in *round*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
round	ObjectType	basic.h

Calls	
Function	**Where Described**
need_to_send_veh_status	Section 2.1.1.3.1.32.1

Table 2.2-155: ammo_add_round Information.

2.2.5.1.42 ammo_subtract_round

This routine decrements the ready rack quantity of the type of round passed in *round*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
round	ObjectType	basic.h

Calls	
Function	**Where Described**
need_to_send_veh_status	Section 2.1.1.3.1.32.1

Table 2.2-156: ammo_subtract_round Information.

2.2.5.1.43 ammo_blast_door_open

This routine returns TRUE if the blast door is open.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	char	the blast door is open
FALSE	char	the blast door is closed

Table 2.2-157: ammo_blast_door_open Information.

2.2.5.1.44 ammo_turret_power_on

This routine sets the turret power status to ON_EDGE. When turret power is turned on, the turret power status stays in ON_EDGE until all subsystems are completely powered.

2.2.5.1.45 ammo_turret_power_off

This routine sets the turret power status to OFF_EDGE. When turret power is turned off, the turret power status stays in OFF_EDGE until all subsystems are completely shut down.

2.2.5.1.46 ammo_breech_ready

This routine determines if the breech is ready to be loaded. The loader timer must have timed out this tick, the loaders arms must be full, the breech must be empty, and the ejection guard must be open. If all the conditions are met, the routine returns TRUE. Otherwise, the routine returns FALSE.

Return Values		
Return Value	**Type**	**Meaning**
(timers_get_timeout_edge (loader_timer_number)) && (loaders_arms!=EMPTY) && (breech_ammo_type == EMPTY) && (ejection_guard_status == SAFE)	char	If all conditions are met, TRUE is returned, signifying that the breech is ready for loading; If any condition is not met, FALSE is returned, signifying that the breech is not ready for loading.

Calls	
Function	**Where Described**
timers_get_timeout_edge	2.6.3.22.1

Table 2.2-158: ammo_breech_ready Information.

2.2.5.1.47 ammo_start_loader_timer

This routine starts the loader timer. The loader timer is set for the minimum time it would take the loader to carry the shell. If the loader timer was timing already, it is restarted by replacing it with a new timer and freeing the old timer.

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

Table 2.2-159: ammo_gun_fired Information.

2.2.5.1.48 ammo_stop_loader timer

This routine stops the loader timer and frees up its timer.

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

Table 2.2-160: ammo_stop_loader_timer Information.

2.2.5.1.49 ammo_start_blast_door_timer

This routine starts the blast door timer. The blast door timer is set for the amount of time the blast door stays open. If the loader timer was timing already, it is restarted by replacing it with a new timer and freeing the old timer.

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

Table 2.2-161: ammo_start_blast_door_timer Information.

2.2.5.1.50 ammo_stop_blast_door_timer

This routine stops the blast door timer, frees up its timer, and closes the blast door.

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

Table 2.2-162: ammo_stop_blast_door_timer Information.

2.2.5.1.51 ammo_blast_door_check

This routine checks the blast door timer each tick. If the timer timed out, the timer is freed, the blast door is closed, and the hydraulic ammo door is closed.

Calls	
Function	**Where Described**
timers_get_timeout_edge	Section 2.6.3.22.1
timers_free_timer	Section 2.6.3.5.1
timers_set_null_timer	Section 2.6.3.14.1
ammo_close_blast_door	Section 2.2.5.1.30
hydraulic_ammo_door_closed	Section 2.2.6.4.2.8

Table 2.2-163: ammo_blast_door_check Information.

2.2.5.1.52 ammo_flash_check

This routine checks the status of resupplies every tick for the purpose of flashing the resupply lamps. If either a resupply receive timer is still timing, or a resupply send is in progress, the routine will flash the resupply lamps to indicate the source and destination of the transfer. When the resupply is complete, the flashing is ended.

Calls	
Function	**Where Described**
controls_resupply_flash	Section 2.2.2
controls_resupply_unflash	Section 2.2.2

Table 2.2-164: ammo_flash_check Information.

2.2.5.1.53 ammo_resupply_receive_timeout_check

This routine checks the status of any resupply receive every tick. If the resupply receive timer timed out this tick, the timer is freed. The resupply status is checked.

In the case of an internal resupply, the ammo is transferred to an empty ready rack slot making the appropriate sound. The slot's ammo lamp is turned on. The original location of the round's rack quantity is decremented, and the ready rack quantity of the round is incremented. The resupply status, location, and slot flags are reset, and a vehicle status message is sent, indicating the changes in ammo distribution within the tank.

In the case of an external resupply receive, the appropriate ammo type is received in the appropriate rack, and the rack quantity is incremented. If ammo is received in the ready rack, the sound of the ready rack insert is made and the slot's ammo lamp is turned on. A vehicle status message is sent, indicating the changes in ammo distribution within the tank.

Errors	
Error	**Reason for Error**
stderr AMMO: ammo_resupply_receive_timeout_check	- impossible resupply location - impossible resupply status

Calls	
Function	**Where Described**
timers_get_timeout_edge	Section 2.6.3.22.1
timers_free_timer	Section 2.6.3.5.1
timers_set_null_timer	Section 2.6.3.14.1
sound_make_const_sound	Section 2.1.3.1.2
controls_show_round	Section 2.2.2
ammo_add_round	Section 2.2.5.1.41
controls_resupply_empty	Section 2.2.2
need_to_send_veh_status	Section 2.1.1.3.1.32.1
controls_resupply_restore	Section 2.2.2
resupply_ammo_received	Section 2.2.5.3.10

Table 2.2-165: ammo_resupply_receive_timeout_check Information.

2.2.5.1.54 ammo_start_resupply_receive_timer

This routine starts the resupply receive timer. This timer is set for the time it takes to transfer the ammo to the receiver in an external resupply. If the resupply receive timer was timing already, it is restarted by replacing it with a new timer and freeing the old timer.

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

Table 2.2-166: ammo_start_resupply_receive_timer Information.

2.2.5.1.55 ammo_stop_resupply_receive_timer

This routine stops the resupply receive timer, freeing up the timer.

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

Table 2.2-167: ammo_stop_resupply_receive_timer Information.

2.2.5.1.56 ammo_change_resupply

This routine changes the resupply slot to *slot*.

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

Calls	
Function	**Where Described**
controls_resupply_restore	Section 2.2.2

Table 2.2-168: ammo_change_resupply Information.

2.2.5.1.57 ammo_stop_resupply

This routine stops the resupply. All timers are stopped, and the resupply status and resupply location are set to NO_TRANSFER_VAL.

Calls	
Function	**Where Described**
ammo_stop_resupply_receive_timer	Section 2.2.5.1.55
controls_resupply_restore	Section 2.2.2
resupply_stop_ammo_resupply	Section 2.2.5.30

Table 2.2-169: ammo_stop_resupply Information.

2.2.5.1.58 ammo_start_internal_resupply

This routine starts an internal resupply transfer of ammo. *slot* is the resupply slot which is filled in the transfer.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
slot	int	Standard
Calls		
Function	**Where Described**	
ammo_start_resupply_receive_timer	Section 2.2.5.1.54	
controls_resupply_restore	Section 2.2.2	

Table 2.2-170: ammo_start_internal_resupply Information.

2.2.5.1.59 ammo_start_external_resupply

This routine starts the external resupply of ammo. *heat_offered* and *apds_offered* are the quantities of heat and apds rounds offered by the resupply sender. This routine calls **ammo_decide_resupply_receive** in order to determine the amount of each ammo type to receive and where to place it. If the desired location is unavailable, the routine returns FALSE. Otherwise, the resupply is started and the routine returns TRUE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
heat_offered	int	Standard
apds_offered	int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
round_location	char	Standard
round_slot	char	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	Unsuccessful resupply; the round location is unavailable.
FALSE	int	Successful resupply.
Calls		
Function	**Where Described**	
ammo_decide_resupply_receive	Section 2.2.5.1.61	
ammo_start_resupply_timer	Section 2.2.5..1.51	
controls_resupply_restore	Section 2.2.2	

Table 2.2-171: ammo_start_external_resupply Information.

2.2.5.1.60 ammo_start_external_send

This routine start an external transfer send.

Calls	
Function	**Where Described**
controls_resupply_restore	Section 2.2.2

Table 2.2-172: ammo_start_external_send Information.

2.2.5.1.61 ammo_decide_resupply_receive

This routine determines the quantity of each ammo type to receive and where to put it. *heat_offered* and *apds_offered* are the quantities of heat and apds rounds offered by the sender; *location_ptr* is filled in with the rack in which to receive the next round; *slot_ptr* is filled in with an empty slot in which to place the next round.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
heat_offered	int	Standard
apds_offered	int	Standard
slot_ptr	pointer to int	Standard
location_ptr	pointer to char	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
round_type	ObjectType	basic.h
Calls		
Function	**Where Described**	
ammo_decide_receive_location	Section 2.2.5.1.62	
ammo_decide_round_type	Section 2.2.5.1.63	

Table 2.2-173: ammo_decide_resupply_receive Information.

2.2.5.1.62 ammo_decide_receive_location

This routine determines the rack and slot in which to put the round. *round_type* is the round to be placed; *location_ptr* is filled in with the rack in which to receive the round; *slot_ptr* is filled in with an empty slot in which to place the round

Parameters		
Parameter	**Type**	**Where Typedef Declared**
round_type	ObjectType	basic..h
location_ptr	pointer to char	Standard
slot_ptr	pointer to int	Standard

Calls	
Function	**Where Described**
ammo_decide_resupply_slot	Section 2.2.5.1.68

Table 2.2-174: ammo_decide_receive_location Information.

2.2.5.1.63 ammo_decide_round_type

This routine determines which round type is needed most.

Return Values		
Return Value	**Type**	**Meaning**
munition_US_M456A1	ObjectType	Heat round.
munition_US_M392A2	ObjectType	Apds round

Table 2.2-175: ammo_decide_round_type Information.

2.2.5.1.64 ammo_stop_timers

This routine stops and frees all timers: the loader timer, the blast door timer, and the resupply receive timer.

Calls	
Function	**Where Described**
timers_free_tiemr	Section 2.6.3.5.1
timers_set_null_timers	Section 2.6.3.14.1

Table 2.2-176: ammo_stop_timers Information.

2.2.5.1.65 ammo_restore_ammo

This routine restores all ammo. It stops any resupply in progress, empties the loaders arms, empties the breech, and initializes the ammo racks.

Internal Variables		
Variable	Type	Where Typedef Declared
i	int	Standard

Calls	
Function	Where Described
ammo_stop_resupply	Section 2.2.5.1.57
controls_show_breech	Section 2.2.2
ammo_init_ammo_racks	Section 2.2.5.1.5
controls_show_round	Section 2.2.2

Table 2.2-177: ammo_restore_ammo Information.

2.2.5.1.66 ammo_resupply_sent

This routine is only used by the keyboard.

Parameters		
Parameter	Type	Where Typedef Declared
ammo_type	int	Standard

Errors	
Error	Reason for Error
stderr AMMO:P ammo_resupply_sent	impossible resupply location

Calls	
Function	Where Described
ammo_decide_resupply_send	Section 2.2.5.1.67
controls_resupply_empty	Section 2.2.2
need_to_send_veh_status	Section 2.1.1.3.1.32.1
sound_make_const_sound	Section 2.1.3.1.2
controls_unshow_round	Section 2.2.2
ammo_subtract_round	Section 2.2.5.1.42
controls_resupply_restore	Section 2.2.2

Table 2.2-178: ammo_resupply_sent Information.

2.2.5.1.67 ammo_decide_resupply_send

This routine determines whether the ammunition passed in *ammo_type* is available to send in an external resupply.

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

Errors	
Error	**Reason for Error**
stderr AMMO: ammo_decide_resupply_send	- no available heat - no available apds - impossible ammo type

Calls	
Function	**Where Described**
ammo_decide_resupply_slot	Section 2.2.5.1.68

Table 2.2-179: ammo_decide_resupply_send Information.

2.2.5.1.68 ammo_decide_resupply_slot

This routine finds the first slot in the ready rack containing the ammo passed in *ammo_type*.

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

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

Return Values		
Return Value	**Type**	**Meaning**
NULL_SLOT	int	all slots are empty or contain wrong ammo
i	int	first slot in ready rack containing appropriate ammo

Table 2.2-180: ammo_decide_resupply_slot Information.

2.2.5.1.69 ammo_print_statistics

This routine prints ammunition statistics for each rack and ammunition type.

2.2.5.1.70 ammo_enable_autoloader

This routine enables the autoloader. The autoloader is enabled with the command line -A in order to perform ammo transfers without a person acting as loader.

2.2.5.2 m1_fuelsys.c

(./simnet/release/src/vehicle/m1/src/m1_fuelsys.c [m1_fuelsys.c])

This CSU keeps track of the M1's fuel supply.

Includes:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libfail.h"
"failure.h"
"m1_fuelsys.h"
"m1_fuel_df.h"
"m1_ctrl.h"
"m1_meter.h"
"m1_engine.h"
"timers.h"
"timers_dfn.h"
"m1_resupp.h"

Declarations:

fuel_warning_levels()
fuel_meter_value()
fuel_check_xfer_time()
fuel_xfer_fuel()
fuel_rear_tank_not_empty()
fuel_resupply_tank()
rear_fuel_level
r_front_fuel_level
l_front_fuel_level
fuel_xfer_ok -- NO_XFER = no transfer, TIMER_ON = on timer, OK = can transfer
xfer_timer_id
fuel_level -- LOW or NORMAL
tank_selected -- REAR, L_FRONT, or R_FRONT
xfer_status -- ON = transferring fuel, OFF = dormant
fuelsys_status -- WORKING or BROKEN
xfer_pump_status -- WORKING or BROKEN
fuel_flow -- fuel usage rate in gallons per hour
resupply_timer_id -- The timer id for resupply
tank_being_resupplied -- The tank which is receiving fuel
total_resupplied -- The fuel taken in a resupply interval
resupply_status -- If TRUE, a resupply is in progress
mcc_offering -- The amount of fuel offered by the MCC

Defines:

RESUPPLY_INTERVAL -- Interval (seconds) before sending reply to the MCC
RESUPPLY_RATE -- GPM of fuel transfer from truck
FUEL_PER_INTERVAL

2.2.5.2.1 fuel_init_tanks

This routine is used by the MCC to intitialize the fuel level in each tank. This routine should be called before **fuel_init**().

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rear	REAL	sim_types.h
l_front	REAL	sim_types.h
bottomr_front	REAL	sim_types.h

Table 2.2-181: fuel_init_tanks Information.

2.3.5.2.2 fuel_init

This routine initializes the fuel system. The transfer status variables, resupply status variables, and failures are inititalized.

Calls	
Function	**Where Described**
fail_init_failure	Section 2.5.4.11.2

Table 2.2-182: fuel_init Information.

2.2.5.2.3 fuel_simul

This routine simulates the various functions of the fuel system on a tick by tick basis. These functions consist of: 1) calculating fuel usage during the current tick, 2) turning on the fuel warning levels if transitioning to the low fuel condition, 3) calculating the levels in each tank based on the fuel usage this tick, 4) transferring fuel when required, 5) resupplying fuel, 6) stopping the fuel resupply, and 7) setting the meter to read the amount of fuel in the tank selected.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
fuel_usage_this_tick	REAL	sim_types.h

Calls	
Function	**Where Described**
fuel_warning_levels	Section 2.2.5.2.5
engine_running	Section 2.2.6.2.2.8
fuel_rear_tank_not_empty	Section 2.2.5.2.7
fuel_check_xfer_timer	Section 2.2.5.2.6
fuel_xfer_fuel	Section 2.2.5.2.12
fuel_resupply_tank	Section 2.2.5.2.24
timers_get_ticking_status	Section 2.6.3.20.1
fuel_stop_resupply	Section 2.2.5.2.23
fuel_meter_value	Section 2.2.5.2.4

Table 2.2-183: fuel_simul Information.

2.2.5.2.4 fuel_meter_value

This routine sets the fuel tank meter to show the fuel level in the tank selected by the tank select switch.

Errors	
Error	**Reason for Error**
stderr	WARNING: invalid tank selected
Calls	
Function	**Where Described**
meter_fuel_set	Section 2.2.2.3.1

Table 2.2-184: fuel_meter_value Information.

2.2.5.2.5 fuel_warning_levels

This routine calls the controls software to turn on the *fuel_level_low* light on when the rear fuel tank is less than or equal to 1/4 full. The routine calls the controls software to turn off the *fuel_level_low* light once the fuel level has reached the 3/8 full level. Fuel cannot be transferred unless the *fuel_level_low* light is on.

Calls	
Function	**Where Described**
controls_low_fuel_off	Section 2.2.2
controls_low_fuel_on	Section 2.2.2

Table 2.2-185: fuel_warning_levels Information.

2.2.5.2.6 fuel_check_xfer_timer

When master power is turned on, there is a 5 second period, counted by the fuel transfer timer, *xfer_timer_id*, during which fuel cannot be transferred. This routine checks to see if the fuel transfer timer has timed out.

Calls	
Function	**Where Described**
timers_get_ticking_status	Section 2.6.3.20.1
timers_free_timer	Section 2.6.3.5.1

Table 2.2-186: fuel_check_xfer_timer Information.

2.2.5.2.7 fuel_rear_tank_not_empty

This routine returns TRUE if the rear fuel tank level is above the minimum fuel level, and FALSE if the level is below the minimum fuel level.

Return Values		
Return Value	**Type**	**Meaning**
rear_fuel_level > MIN_FUEL_LEVEL ? TRUE : FALSE	BOOLEAN	If TRUE, the rear fuel tank level is above the minimum level; If FALSE, the rear fuel tank level is below the minimum level

Table 2.2-187: fuel_rear_tank_not_empty Information.

2.2.5.2.8 fuel_set_flow

This routine sets the *fuel_flow* variable equal to the parameter *value.*

Parameters		
Parameter	**Type**	**Where Typedef Declared**
value	REAL	sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
fuelsys_status && fuel_top_tank_not_empty()	BOOLEAN	If TRUE, then the fuel button is ON, the rear fuel tank is not empty, and the *fuel_flow* variable is set to *value*; If FALSE, then either the fuel button is OFF or the rear fuel tank is empty

Calls	
Function	**Where Described**
fuel_rear_tank_not_empty	Section 2.2.5.2.7

Table 2.2-188: fuel_set_flow Information.

2.2.5.2.9 fuel_select_front_left_tank

This routine is called when the fuel tank selector switch is switched to the left front tank position. The meter will automatically display the fuel in the left front tank. Also, if the fuel level is LOW and the transfer status is OK, a transfer of fuel from the left front tank to the rear tank will commence.

Errors	
Error	**Reason for Error**
stderr	Fuel tranfer pump broken

Table 2.2-189: fuel_select_front_left_tank Information.

2.2.5.2.10 fuel_select_front_right_tank

This routine is called when the fuel tank selector switch is switched to the right front tank position. The meter will automatically display the fuel in the right front tank. Also, if the fuel level is LOW and the transfer status is OK, a transfer of fuel from the right front tank to the rear tank will commence.

Errors	
Error	**Reason for Error**
stderr	Fuel tranfer pump broken

Table 2.2-190: fuel_select_front_right_tank Information.

2.2.5.2.11 fuel_select_rear_tank

This routine is called when the fuel tank selector switch is switched to the rear tank position. When the rear tank is selected, any transfer that has been taking place will be stopped. If the fuel level is NORMAL, no more fuel can be transferred until it becomes LOW.

2.2.5.2.12 fuel_xfer_fuel

This routine calculates the right front, left front, and rear tank levels after a fuel transfer between tanks. If a fuel transfer is in progress, it ends if either 1) the supply tank becomes empty, 2) the selector switch is switched back to rear tank, during which the transfer cannot be resumed unless the fuel low light is still on, or 3) the rear tank reaches 3/4 full.

Errors	
Error	**Reason for Error**
stderr	WARNING: invalid tank selection for transfer

Table 2.2-191: fuel_xfer_fuel Information.

2.2.5.2.13 fuel_master_power_on

This routine is called by the electrical system software to let the fuel system know that the master power has been turned on. A 5 second delay after master power has been turned on is counted by *xfer_timer_id* in order to prevent accidental fuel transfers.

Calls	
Function	**Where Described**
timers_get_timer	Section 2.6.3.6.1

Table 2.2-192: fuel_master_power_on Information.

2.2.5.2.14 fuel_master_power_off

This routine is called by the electrical system software to let the fuel system know that the master power has been turned off. When the master power is turned off, fuel can no longer be transferred.

Calls	
Function	**Where Described**
timers_free_timer	Section 2.6.3.5.1

Table 2.2-193: fuel_engine_accessory_off Information.

2.2.5.2.15 fuel_level_rear

This routine returns the fuel level of the rear fuel tank.

Return Values		
Return Value	**Type**	**Meaning**
rear_fuel_level	REAL	the level in the rear fuel tank

Table 2.2-194: fuel_level_rear Information.

2.2.5.2.16 fuel_level_left

This routine returns the fuel level of the left front fuel tank.

Return Values		
Return Value	**Type**	**Meaning**
l_front_fuel_level	REAL	the level in the left front fuel tank

Table 2.2-195: fuel_level_left Information.

2.2.5.2.17 fuel_level_right

This routine returns the fuel level of the right front fuel tank.

Return Values		
Return Value	**Type**	**Meaning**
r_front_fuel_level	REAL	the level in the right front fuel tank

Table 2.2-196: fuel_level_right Information.

2.2.5.2.18 fuel_repair_transfer_pump

This routine is called by "m1_repairs.c" to repair a broken fuel transfer pump.

Calls	
Function	**Where Described**
controls_right_pump_inoperative_off	Section 2.2.2
controls_left_pump_inoperative_off	Section 2.2.2
fuel_select_front_left_tank	Section 2.2.5.2.9
fuel_select_front_right_tank	Section 2.2.5.2.10

Table 2.2-197: fuel_repair_transfer_pump Information.

2.2.5.2.19 fuel_transfer_pump_failure

This routine is called by the failures module to break the fuel transfer pump.

Errors	
Error	**Reason for Error**
stderr	DAMAGE: fuel transfer pump failure
Calls	
Function	**Where Described**
controls_right_pump_inoperative_on	Section 2.2.2
controls_left_pump_inoperative_on	Section 2.2.2

Table 2.2-198: fuel_transfer_pump_failure Information.

2.2.5.2.20 fuel_supply_full

This routine calculates whether the amount of fuel in the parameter *delta* will fill the fuel tanks.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
delta	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	*delta* will fill the fuel supply
FALSE	BOOLEAN	*delta* will not fill the fuel supply

Table 2.2-199: fuel_supply_full Information.

2.2.5.2.21 fuel_decide_resupply_quantity

This routine calculates the amount of fuel needed to fill the three tanks. It returns either the maximum quanity of fuel allowed to transfer in one resupply interval or the actual amount of fuel necessary to fill the tanks (whichever is lower).

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
fuel_needed	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
(int) min(fuel_needed, FUEL_PER_INTERVAL)	int	the quantity of fuel necessary for resupply

Table 2.2-200: fuel_decide_resupply_quantity Information.

2.2.5.2.22 fuel_start_external_resupply

This routine starts the process of externally resupplying fuel. If the amount of fuel the MCC is offering is less than 0 gallons, the routine returns FALSE. Otherwise, the routine starts the resupply timer, determines which tanks should be resupplied, and returns TRUE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fuel_offered	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	the external resupply was started
FALSE	BOOLEAN	the external resupply was not started
Calls		
Function	**Where Described**	
timers_get_timer	Section 2.6.3.6.1	

Table 2.2-201: fuel_start_external_resupply Information.

2.2.5.2.23 fuel_stop_resupply

This routine stops the fuel resupply. The timers are freed, the resupply status is set to OFF, and the amount of fuel received during the resupply is calculated.

Calls	
Function	**Where Described**
timers_free_timers	Section 2.6.3.5.1
resupply_fuel_received	Section 2.2.5.3.11

Table 2.2-202: fuel_stop_resupply Information.

2.2.5.2.24 fuel_resupply_tank

This routine calculates the amount of fuel in each tank after a fuel resupply. If the fuel transfer is in progress, it ends if either 1) the supply tank becomes empty, 2) the selector switch is turned back to the rear tank, during which the transfer cannot be resumed unless the fuel low light is still on, or 3) the rear tank level reaches 3/4 full.

Errors	
Error	**Reason for Error**
stderr	WARNING: invalid tank selection for resupply

Table 2.2-203: fuel_resupply_tank Information.

2.2.5.3 m1_resupp.c
(./simnet/release/src/vehicle/m1/src/m1_resupp.c [m1_resupp.c])

Resupply of fuel and ammunition is coordinated by this CSU.

This files includes:
"stdio.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"timers_dfn.h"
"timers.h"
"mun_type.h"
"libnetwork.h"
"pro_sim.h"
"m1_ammo_df.h"
"m1_ammo_mx.h"
"m1_ammo.h"
m1_cntrl.h"
"m1_tracks.h"
"m1_repair.h"
"m1_fuel_df.h"
"pro_assoc.h"
"assoc.h"

Defines:
QUIET
REQUEST
LOADING
WAITING
SERVICING
MAX_SERVICE_ENTITIES

Declarations:
ammo_carrier
fuel_carrier
ammo_receiver
num_ammo_carriers
num_fuel_carriers
num_ammo_receivers
ammo_carriers_near_here
fuel_carriers_near_here
ammo_receivers_near_here

ammo_resupply_receive_state
ammo_has_been_received
ammo_that_was_received
ammo_receive_timer_id
ammo_carrier_id

fuel_resupply_receive_stae
fuel_has_been_received
fuel_that_was_received
fuel_receive_timer_id
fuel_carrier_id
ammo_resupply_send_state
ammo_send_timer_id
ammo_receiver_id

clear_ammo_carriers()
clear_fuel_carriers()
clear_ammo_receivers()
send_feed_me_packets_ammo_carriers()
send_feed_me_packets_fuel_carriers()
ammo_receive_quiet_state()
fuel_receive_quiet_state()
ammo_send_quiet_state()
ammo_receive_request_state()
fuel_receive_request_state()
ammo_send_waiting_state()
ammo_receive_loading_state()
fuel_receive_loading_state()
ammo_send_servicing_state()
ammo_resupply_receive_simul()
fuel_resupply_receive_simul()
ammo_resupply_send_simul()
vehicle_is_close()

2.2.5.3.1 clear_ammo_carriers

This routine clears the *ammo_carriers_near_here* flag to FALSE and the number of ammo carriers to zero.

2.2.5.3.2 clear_fuel_carriers

This routine clears the *fuel_carriers_near_here* flag to FALSE and the number of fuel carriers to zero.

2.2.5.3.3 clear_ammo_receivers

This routine clears the *ammo_carriers_near_here* flag to FALSE and the number of ammo receivers to zero.

2.2.5.3.4 print_resupply_status

This routine prints the resupply status information.

2.2.5.3.5 send_feed_me_packets_ammo_carriers

This routine sends a service request packet to each of the ammo carriers on the network, requesting the specific types of ammunition necessary for resupply.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
munition	MunitionQuantity	basic.h

Calls	
Function	**Where Described**
ammo_decide_round_type	Section 2.2.5.1.63
network_send_feed_me_packet	Section 2.1.1.3.1.28.1

Table 2.2-204: send_feed_me_packets_ammo_carriers Information.

2.2.5.3.6 send_feed_me_packets_fuel_carriers

This routine sends a service request packet to each of the fuel carriers on the network, requesting a specific quantity of fuel necessary for resupply.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
munition	MunitionQuantity	basic.h

Calls	
Function	**Where Described**
fuel_decide_resupply_quantity	Section 2.2.5.2.21
network_send_feed_me_packet	Section 2.1.1.3.1.48.1

Table 2.2-205: send_feed_me_packets_fuel_carriers Information.

2.2.5.3.7 resupply_near_ammo_carrier

This routine maintains the list of close vehicles which are ammo carriers. If any ammo carriers are on this list, the *ammo_carriers_near_here* flag is set to TRUE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v	pointer to VehicleID	basic.h

Table 2.2-206: resupply_near_ammo_carrier Information.

2.2.5.3.8 resupply_near_fuel_carrier

This routine maintains the list of close vehicles which are fuel carriers. If any fuel carriers are on this list, the *fuel_carriers_near_here* flag is set to TRUE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v	pointer to VehicleID	basic.h

Table 2.2-207: resupply_near_fuel_carrier Information.

2.2.5.3.9 resupply_near_ammo_receiver

This routine maintains the list of close vehicles which are ammo receivers. If any ammo receivers are on this list, the *ammo_receivers_near_here* flag is set to TRUE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v	pointer to VehicleID	basic.h

Table 2.2-208: resupply_near_ammo_receiver Information.

2.2.5.3.10 resupply_ammo_received

This routine sets the *ammo_has_been_received* flag to TRUE, and set the variable *ammo_that_was_received* to the quantities and types of ammunition that were received.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_type	ObjectType	basic.h

Table 2.2-209: resupply_ammo_received Information.

2.2.5.3.11 resupply_fuel_received

This routine sets the *fuel_has_been_received* flag to TRUE and sets the variable *fuel_that_was_received* equal to the number of gallons of fuel that were received.

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

Table 2.2-210: resupply_fuel_received Information.

2.2.5.3.12 resupply_offer_packet

This routine is called by the LibRcvNet routine **process_resupply_offer**() in order to process a message from a vehicle within range offering munitions resupply. **resupply_offer_packet**() first determines the types and quantities of munitions offered, and then checks the ammo and fuel resupply receive states

If the resupply receive state is QUIET, no munitions have been requested and no munitions are received. If the state is REQUEST, the resupply timer is started, the external resupply of either fuel or ammunition (heat105 or apds105) is started, and the state is changed to LOADING. If the state is LOADING, the external resupply is in progress. If the receive state is not known, an error is printed. Parameters are represented as follows:

carrier_id -- The VehicleID of the munitions carrier.
num_munitions -- The number of munitions types carried.
munitions -- The quantities and types of munitions being carried.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
carrier_id	pointer to VehicleID	basic.h
num_munitions	usigned char	Standard
munitions	register pointer to MunitionQuantity	basic.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
fuel	int	Standard
apds105	int	Standard
heat105	int	Standard
mun_type	ObjectType	basic.h
mun_quantity	float	Standard
i	int	Standard

Errors	
Error	**Reason for Error**
RESUPPLY: resupply_offer_packet	- unkown ammo state - unknown fuel state
Calls	
Function	**Where Described**
timers_free_timer	Section 2.6.3.5.1
ammo_start_external_resupply	Section 2.2.5.1.59
timers_get_timer	Section 2.6.3.6.1

Table 2.2-211: resupply_offer_packet Information.

2.2.5.3.13 resupply_thank_you_packet

This routine is called by the LibRcvNet routine **process_resupply_received**() in order to process a message from a resupply receiver saying that the ammo was received. **resupply_thank_you_packet**() calculates the types and quantities of ammo that were sent to the receiver, frees the resupply timer, sets the resupply send state to QUIET, and stops the ammo resupply.

receiver_id -- The Vehicle ID of the vehicle which received the ammo resupplies.
num_munitions -- The number of different types of ammo sent by the carrier.
munitions -- The quantity of each type of ammo sent by the carrier.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
receiver_id	pointer to VehicleID	basic.h
num_munitions	unsigned char	Standard
munitions	register pointer to MunitionQuantity	basic.h

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

Calls	
Function	**Where Described**
ammo_resupply_sent	Section 2.2.5.1.66
timers_free_timers	Section 2.6.3.5.1
ammo_stop_resupply	Section 2.2.5.1.57

Table 2.2-212: resupply_thank_you_packet Information.

2.2.5.3.14 resupply_feed_me_packet

This routine is called by the LibRcvNet routine **process_service_request**() in order to process a message requesting ammunition resupply from a vehicle within range. **resupply_feed_me_packet**() first checks the ammo resupply send state. If the state is QUIET, no supplies are to be sent. If the state is WAITING, send an offer packet on the network to the receiver listing the types and quantities of munitions that you have, start the external resupply, and change the state to SERVICING. If the state is SERVICING, the external resupply is in progress.

receiver_id -- The VehicleID of the munitions receiver.
num_munitions -- The number of ammo types requested by the receiver.
feed_me_munitions -- The types and quantities of ammo being requested by the reciever.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
receiver_id	pointer to VehicleID	basic.h
num_munitions	unsigned char	Standard
feed_me_munitions	pointer to MunitionQuantity	basic.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
offer_munitions	pointer to MunitionQuantity	basic.h

Errors	
Error	**Reason for Error**
RESUPPLY: resupply_feed_me_packet	unknown ammo
Calls	
Function	**Where Described**
timers_free_timers	Secttion 2.6.3.5.1
timers_get_timer	Secttion 2.6.3.6.1
network_send_offer_packet	Section 2.1.1.3.1.40.1
ammo_start_external_send	Section 2.2.5.1.60

Table 2.2-213: resupply_feed_me_packet Information.

2.2.5.3.15 resupply_gating_conditions

This routine returns TRUE if the vehicle is not moving and there are no failures in controls. The routine returns FALSE if the vehicle is moving or there is a failure in controls.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
tracks_speed	REAL	sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	the vehicle is not moving; no failures in the controls
FALSE	BOOLEAN	either the vehicle is moving or there is a controls failure

Calls	
Function	**Where Described**
drivetrain_get_vehicle_speed	Section 2.2.6.2.1
controls_failure_status	Section 2.2.2

Table 2.2-214: resupply_gating_conditions Information.

2.2.5.3.16 ammo_receive_quiet_state

This routine determines the receiver's ammunition resupply Finite State Machine's QUIET state. If the following conditions are ALL TRUE:

- The resupply gating conditions are TRUE (the vehicle is alive, the vehicle is not moving, and no controls failures exist).
- There are ammo carriers nearby.
- The commander has set the ammunition transfer switch to REDIST_RECV.
- The ammunition load in the tank is not full.
- The loader's arms are empty.

Then, send a feed me packet to the ammo carriers on the network and enter the REQUEST state. If any of the conditions are not met, remain in the QUIET state.

Return Values		
Return Value	**Type**	**Meaning**
REQUEST	int	the receiver is in the REQUEST state
QUIET	int	the receiver is in the QUIET state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.2.5.3.15
ammo_decide_round_type	Section 2.2.5.1.63
launcher_get_val	
timers_get_timer	Section 2.6.3.6.1
send_feed_me_packet_ammo_carriers	Section 2.2.5.3.5

Table 2.2-215: ammo_receive_quiet_state Information.

2.2.5.3.17 fuel_receive_quiet_state

This routine determines the receiver's fuel resupply Finite State Machine's QUIET state. If the following conditions are ALL TRUE:

- The resupply gating conditions are TRUE (the vehicle is alive, the vehicle is not moving, and no controls failures exist).
- There are fuel carriers nearby.
- There is room for the fuel.

Then, send a feed me packet to the fuel carriers on the network, start the resupply timer, and enter the REQUEST state. If any of the conditions are not met, remain in the QUIET state.

Return Values		
Return Value	**Type**	**Meaning**
REQUEST	int	the receiver is in the REQUEST state
QUIET	int	the receiver is in the QUIET state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.2.5.3.15
fuel_supply_full	Section 2.2.5.2.20
timers_get_timer	Section 2.6.3.6.1
send_feed_me_packets_fuel_carriers	Section 2.2.5.3.5

Table 2.2-216: fuel_receive_quiet_state Information.

2.2.5.3.18 ammo_send_quiet_state

This routine determines the sender's ammunition resupply Finite State Machine's QUIET state. If the following conditions are ALL TRUE:

- The resupply gating conditions are TRUE (the vehicle is alive, the vehicle is not moving, and no controls failures exist).
- There are ammo receivers nearby.
- The commander has set the ammunition transfer switch to REDIST_SEND_VAL.
- The ammunition load in the tank is not empty.
- The loader's arms are empty.

Then, enter the WAITING state. If any conditions are not met, remain in the QUIET state.

Return Values		
Return Value	**Type**	**Meaning**
WAITING	int	the sender is in the WAITING state
QUIET	int	the sender is in the QUIET state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.2.5.3.15

Table 2.2-217: ammo_send_quiet_state Information.

2.2.5.3.19 ammo_receive_request_state

This routine determines the receiver's ammunition resupply Finite State Machine's REQUEST state. If ANY of the following conditions are TRUE:

- The resupply gating conditions are FALSE (the vehicle is dead, the vehicle is moving, or a controls failure exists).
- There are no ammo carriers nearby.
- The commander has not set the ammunition transfer switch to REDIST_RECV_VAL.
- The ammunition supply is full.
- The loader's arms are full.

Then, abort the resupply timer and enter the QUIET state. If none of the conditions are met and the resupply timer has expired, send another service request, restart the timer, and remain in the REQUEST state. If the resupply timer has not expired, remain in the REQUEST state.

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	the receiver is in the QUIET state
REQUEST	int	the receiver is in the REQUEST state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.2.5.3.15
timers_get_timeout_edge	Section 2.6.3.22.1
timers_free_timer	Section 2.6.3.5.1
timers_get_timer	Section 2.6.3.6.1
send_feed_me_packets_ammo_carriers	Section 2.2.5.3.5

Table 2.2-218: ammo_receive_request_state Information.

2.2.5.3.20 fuel_receive_request_state

This routine determines the receiver's fuel resupply Finite State Machine's REQUEST state. If ANY of the following conditions are TRUE:

- The resupply gating conditions are FALSE (the vehicle is dead, the vehicle is moving, or a controls failure exists).
- There are no fuel carriers nearby.
- The fuel supply is full.

Then, abort the resupply timer and enter the QUIET state. If none of the conditions are met and the resupply timer has expired, send another service request, restart the timer, and remain in the REQUEST state. If the resupply timer has not expired, remain in the REQUEST state.

Return Values		
Return Value	**Type**	**Meaning**
REQUEST	int	the receiver is in the REQUEST state
QUIET	int	the receiver is in the QUIET state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.2.5.3.15
fuel_supply_full	Section 2.2.5.2.20
timers_free_timer	Section 2.6.3.5.1
timers_get_timeout_edge	Section 2.6.3.22.1
timers_get_timer	Section 2.6.3.6.1
send_feed_me_packets_fuel_carriers	Section 2.2.5.3.5

Table 2.2-219: fuel_receive_request_state Information.

2.2.5.3.21 ammo_send_waiting_state

This routine determines the sender's ammunition resupply Finite State Machine's WAITING state. If ANY of the following conditions are TRUE:

- The resupply gating conditions are FALSE (the vehicle is dead, the vehicle is moving, or a controls failure exists).
- There are no ammo receivers nearby.
- The commander has not set the ammunition transfer switch to REDIST_SEND_VAL.
- The ammunition supply is empty.
- The loader's arms are full.

Then, enter the QUIET state. If none of the conditions are met remain in the WAITING state.

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	the sender is in the QUIET state
WAITING	int	the sender is in the WAITING state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.2.5.3.15

Table 2.2-220: ammo_send_waiting_state Information.

2.2.5.3.22 ammo_receive_loading_state

This routine determines the receiver's ammunition resupply Finite State Machine's LOADING state. If the ammo has been received, a thank you packet is sent by the receiver listing the type and amount of ammunition taken, and the receiver enters the QUIET state.

If ANY of the following conditions have changed to TRUE in the LOADING state:

- The resupply gating conditions are FALSE (the vehicle is dead, the vehicle is moving, or a controls failure exists).
- There are no ammo carriers nearby.
- The ammo carrier which offered ammunition is dead.
- the commander has not set the ammunition transfer switch to REDIST_RECV_VAL.
- The loader's arms are full.

Then, stop the resupply and enter the QUIET state.

If the loading has not completed, remain in the LOADING state.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
munitions	MunitionQuantity	basic.h

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	in QUIET staate
LOADING	int	in LOADING state

Errors	
Error	**Reason for Error**
AMMO: ammo_receive_loading_state	impossible ammo_that_was_received

Calls	
Function	**Where Described**
network_send_thank_you_packet	Section 2.1.1.3.1.41
resupply_gating_conditions	Section 2.2.5.3.15
vehicle_is_close	Section 2.2.5.3.34
ammo_stop_resupply	Section 2.2.5.1.57

Table 2.2-221: ammo_receive_loading_state Information.

2.2.5.3.23 fuel_receive_loading_state

This routine determines the receiver's fuel resupply Finite State Machine's LOADING state. If the fuel has been received, a thank you packet listing the quantity of fuel taken is sent by the receiver, the fuel light stops flashing, and the receiver enters the QUIET state.

If ANY of the following conditions have changed to TRUE in the LOADING state:

- The resupply gating conditions are FALSE (the vehicle is dead, the vehicle is moving, or a controls failure exists).
- There are no fuel carriers nearby.
- The fuel carrier which offered fuel is dead.
- The fuel load is full.

Then, the resupply is stopped, a thank you packet listing the quantity of fuel taken before the resupply was stopped is sent, the fuel light stops flashing, and the receiver enters the QUIET state.

If the loading has not completed, remain in the LOADING state.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
munition	MunitionQuantity	basic.h

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	The vehicle is in the fuel receive quiet state
LOADING	int	The vehicle is in the fuel receive loading state

Calls	
Function	**Where Described**
network_send_thank_you_packet	Section 2.1.1.3.1.41
resupply_gating_conditions	Section 2.2.5.3.15
fuel_supply_full	Section 2.2.5.2.20
vehicle_is_close	Section 2.2.5.3.34
fuel_stop_resupply	Section 2.2.5.2.23

Table 2.2-222: fuel_receive_loading_state Information.

2.2.5.3.24 ammo_send_servicing_state

This routine determines the sender's ammunition resupply Finite State Machine's SERVICING state. If the resupply timer has timed out, stop the resupply and enter the QUIET state. If the resupply timer has not timed out, remain in the SERVICING state.

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	in quiet state.
SERVICING	int	in servicing state.

Calls	
Function	**Where Described**
timers_get_timeout_edge	Section 2.6.3.22.1
timers_free_timers	Section 2.6.3.5.1
ammo_stop_resupply	Section 2.2.5.1.57

Table 2.2-223: ammo_send_servicing_state Information.

2.2.5.3.25 ammo_resupply_receive_simul

This routine runs the ammunition resupply receive simulation. The routine checks the ammo resupply receive state and calls the appropriate routine for that state.

Errors	
Error	**Reason for Error**
REPAIR: ammo_resupply_receive_simul	unknown state

Calls	
Function	**Where Described**
ammo_receive_quiet_state	Section 2.2.5.3.15
ammo_receive_request_state	Section 2.2.5.3.19
ammo_receive_loading_state	Section 2.2.5.3.22

Table 2.2-224: ammo_resupply_receive_simul Information.

2.2.5.3.26 fuel_resupply_receive_simul

This routine runs the fuel resupply receive simulation. The routine checks the fuel resupply receive state and calls the appropriate routine for that state.

Errors	
Error	**Reason for Error**
REPAIR: fuel_resupply_receive_simul	unknown state
Calls	
Function	**Where Described**
fuel_receive_quiet_state	Section 2.2.5.3.17
fuel_receive_request_state	Section 2.2.5.3.20
fuel_receive_loading_state	Section 2.2.5.3.23

Table 2.2-225: fuel_resupply_receive_simul Information.

2.2.5.3.27 ammo_resupply_send_simul

This routine runs the ammunition resupply send simulation. The routine checks the sender's ammo resupply state and calls the appropriate routine for that state.

Errors	
Error	**Reason for Error**
REPAIR: ammo_resupply_send_simul	unknown state
Calls	
Function	**Where Described**
ammo_send_quiet_state	Section 2.2.5.3.18
ammo_send_waiting_state	Section 2.2.5.3.21
ammo_send_servicing_state	Section 2.2..5.3.24

Table 2.2-226: ammo_resupply_send_simul Information.

2.2.5.3.28 resupply_init

This routine initializes the resupply simulation. All ammo and fuel carriers and receivers are cleared, all resupply states are set to QUIET, and all resupply timers are set to NULL.

Calls	
Function	**Where Described**
clear_ammo_carriers	Section 2.2.5.3.1
clear_fuel_carriers	Section 2.2.5.3.3
clear_ammo_receivers	Section 2.2.5.3.2

Table 2.2-227: resupply_init Information.

2.2.5.3.29 resupply_simul

This routine runs the resupply simulations. The routine calls the ammo receive, ammo send and fuel receive simulation routines.

Calls	
Function	**Where Described**
ammo_resupply_receive_simul	Section 2.2.5.3.25
fuel_resupply_receive_simul	Section 2.2.5.3.26
ammo_resupply_send_simul	Section 2.2.5.3.27
clear_ammo_carriers	Section 2.2.5.3.1
clear_fuel_carriers	Section 2.2.5.3.3
clear_ammo_receivers	Section 2.2.5.3.2

Table 2.2-228: resupply_simul Information.

2.2.5.3.29 service_check_vehicle_type

This routine checks the vehicle ID from the *pkt* parameter, determines its vehicle type, and updates the different lists of close vehicles (ammo carriers, fuel carriers, ammo receivers, etc.).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to VehicleAppearanceVariant	p_sim.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
id	pointer to VehicleID	basic.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	procedure was successful

Calls	
Function	**Where Described**
is_fuel_vehicle	Section 2.6.10.7.1
resupply_near_fuel_carrier	Section 2.2.5.3.8
is_repair_vehicle	Section 2.6.10.10.1
repair_near_repair	Section 2.2.4.2.5
is_ammo_vehicle	Section 2.6.10.2.1
is_ammo_carrier	Section 2.6.10.2.2
resupply_near_ammo_carrier	Section 2.2.5.3.7
is_main_battle_tank	Section 2.6.10.9.1
resupply_near_ammo_receiver	Section 2.2.5.3.9
resupply_near_ammo_carrier	Section 2.2.5.3.7

Table 2.2-229: service_check_vehicle_type Information.

2.2.5.3.30 resupply_stop_ammo_resupply

This routine aborts the ammo resupply simulation, resetting the ammo resupply send (or receive) state to QUIET and freeing the resupply timer.

Calls	
Function	**Where Described**
timers_free_timer	Section 2.6.3.5.1

Table 2.2-230: resupply_stop_ammo_resupply Information.

2.2.5.3.31 resupply_stop_fuel_resupply

This routine aborts the fuel resupply simulation, resetting the fuel resupply receive state to QUIET and freeing the resupply timer.

Calls	
Function	**Where Described**
timers_free_timer	Section 2.6.3.5.1

Table 2.2-231: resupply_stop_fuel_resupply Information.

2.2.5.3.32 resupply_offer_canceled

This routine cancels an offer of service from another vehicle.

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

Table 2.2-232: resupply_offer_canceled Information.

2.2.5.3.33 resupply_request_canceled

This routine cancels a request for service.

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

Table 2.2-233: resupply_request_canceled Information.

2.2.5.3.34 vehicle_is_close

This routine determines if a particular vehicle is on the close vehicles list.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
list	register pointer to VehicleID	basic.h
vehicle	register pointer to VehicleID	basic.h
size_of_list	register int	Standard

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	vehicle is near.
FALSE	int	vehicle is not near.

Calls	
Function	**Where Described**
VEHICLE_IDS_EQUAL	sim_macros.h

Table 2.2-234: vehicle_is_close Information.

2.2.6 M1 Vehicle Model

There are a number of vehicle specific simulation functions. Code is required for modeling the relevant moving elements of a vehicle. It is necessary to simulate the forces applied to the vehicle by its propulsion and suspension systems. The generation and use of electric and hydraulic power is simulated, as are the effects of the user's actions on the visual displays. This CSC is broken down into the following CSC's:

Internal Kinematics
Propulsion Simulation
Vehicle Subsystems

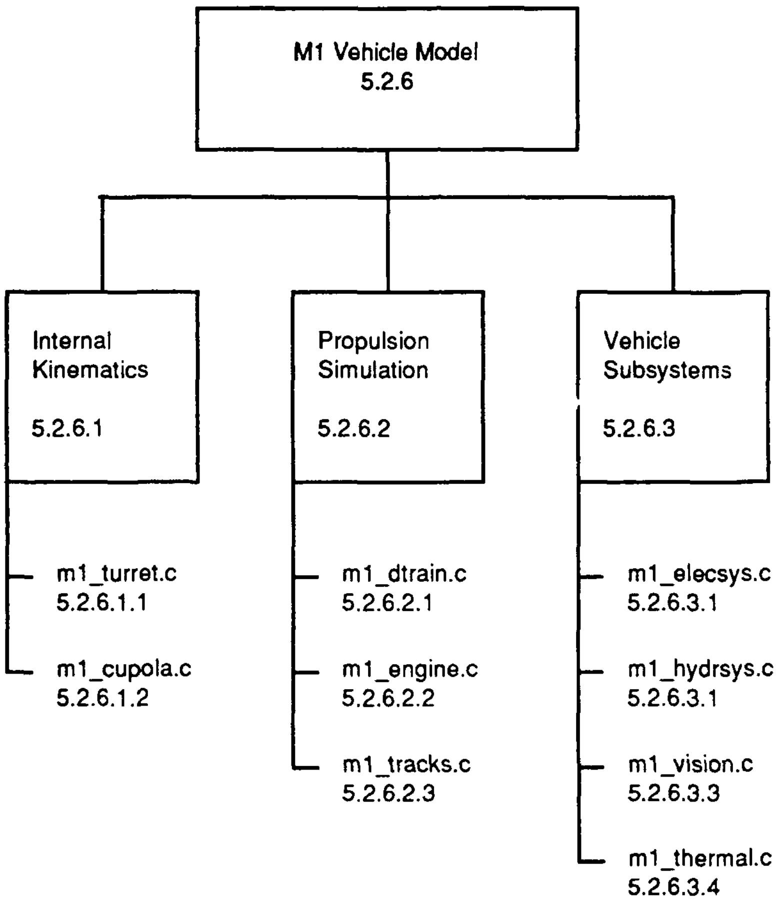

Figure 2.2-7: Structure of the M1 Vehicle Model CSC.

2.2.6.1 Internal Kinematics

The M1 has components which move with respect to the hull. It is necessary to model the movement of these components; however, the required level of model fidelity varies among them. The simulation of moving components on the M1 is accomplished with the following CSU's:

m1_turret.c
m1_cupola.c

2.2.6.1.1 m1_turret.c

(./simnet/release/src/vehicle/m1/src/m1_turret.c [m1_turret.c])

This is the Turret simulation module. It includes the Stabalization System and Gunner's Primary Signt (GPS).

The vehicle specific characteristics of the turrets are modeled in m1_turret.c. The vehicle specific code responds to commands from the controls and determines the appropriate commands to send to libturret. The response depends on the mode of operation of the turret and is affected if a turret subsystem has failed. The operational modes and failure status are tracked in vehicle specific code. Functions are provided to send commands to the sound system provide information to other components of the simulation. This information is needed by the CIG and by some control displays.

The design of the Delta Graphics system fixed reticle changes the relationship between the GPS, reticle, and gun in the simulated M1 tank from a real M1 tank. In a real M1, if the turret is not moving, or if the turret is only moving because of the stab system, then the GPS, reticle, and gun all point in the same direction. However, if the turret is moving because of the gunner's or commander's handles, then the reticle and GPS will be displaced from the gun tube as follows: the first 30 mils of lead tracking will be compensated for by the GPS only, so that the reticle remains centered in the GPS. Beyond 30 mils of lead tracking, the reticle is displaced within the sight. However, because of the restrictions placed by the graphics box, the M1 simulator contains certain compromises. The GPS always points the same way as the turret, whether or not any lead tracking needs to be performed. All lead tracking is performed by displacing the gun from the GPS.

Includes:

```
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"pro_data.h"
"libturret.h"
"libmatrix.h"
"libkin.h"
"libhull.h"
"timers.h"
"timers_dfn.h"
"libfail.h"
"failure.h"
"libsound.h"
"m1_bcs.h"
"m1_weapons.h"
"m1_hydrsys.h"
"m1_sound.h"
"m1_sound_dfn.h"
"m1_tracks.h"
"m1_turret.h"
```

Defines:

TURRET_DEBUG	
TURRET_FAILURES_DEBUG	
STAB_DEBUG	
GUN_BREAK_SPEED	miles/hr
SLEW_ELEC_SLOPE	slew rate coeff
SLEW_ELEC_INT	slew rate coeff
SLEW_HYDR_A	cubic coeff
SLEW_HYDR_B	cubic coeff
SLEW_HYDR_C	cubic coeff
SLEW_HYDR_D	cubic coeff
SLEW_HYDR_KICK_IN	hydraulic kick in
MIN_ELEC_SLEW_RATE	
ELEV_ELEC_SLOPE	elev rate coeff
ELEV_ELEC_INT	elev rate coeff
ELEV_HYDR_A	cubic coeff
ELEV_HYDR_B	cubic coeff
ELEV_HYDR_C	cubic coeff
ELEV_HYDR_D	cubic coeff
ELEV_HYDR_KICK_IN	Hydraulic kick-in

The following are defined for the gunner's primary sight:

MAX_GPS_ELEV	degrees
MIN_GPS_ELEV	degrees
MAX_GPS_ELEV_SIN	degrees
MIN_GPS_ELEV_SIN	degrees

Declared:

MAX_ELEC_ELEV_RATE	
MAX_ELEC_SLEW_RATE	
gps_rel_heading	
gps_rel_elevation	
gyro_speed	between 0.0 and 1.0
gun_slew_handle,	between -1.0 and 1.0
gun_elev_handle	between -1.0 and 1.0
gps_slew_rate	radians per frame
sin_elev_rads	the number of radians that the gun is elevated (relative to the orientation of the hull)
gun_on_stop	TRUE or FALSE
gearbox_status	all either WORKING or BROKEN
elevation_status	
mount_int_status	
stab_status	
traverse_status	
control_engaged	either ON or OFF
fire_ctl_mode	can be any of these three: FCM_NORMAL FCM_MANUAL FCM_EMERGENCY

gun_turret_drive	can be any of these: GTD_UNCOUPLED GTD_POWERED GTD_MANUAL
gyro_direction;	can be any of these three: GYROS_NOT_CHANGING GYROS_SPOOLING_UP GYROS_SPOOLING_DOWN

It takes 25 seconds for the turret gyros to completely come up to speed -- we also assume that they take 25 seconds to completely spin down when shut off

TOTAL_GYRO_TIME
TOTAL_GYROFRAMES
GYRO_SPOOLUP_RATE

It takes 1.4 seconds to completely elevate or depress the gun.

TOTAL_ELEV_TIME seconds
TOTAL_ELEV_FRAMES
MAX_ELEV_RATE

It takes 9.0 seconds to slew the turret a full 360 degrees.

TOTAL_SLEW_TIME seconds
TOTAL_SLEW_FRAMES
MAX_SLEW_RATE

The following functions are declared:

calc_elev_from_handle();
calc_slew_from_handle();
turret_gyros_simul();
turret_move();
turret_calc_gun_elev();
turret_calc_turret_slew();
make_sound_of_no_turret_noise();
make_sound_of_no_elevating();
make_sound_of_no_slewing();

2.2.6.1.1.1 turret_init

This routine initializes the turret variables. The stab vectors are also initialized in order to use the stabilization system.

Calls	
Function	**Where Described**
turret_set_stab_sys	Section 2.5.5.2.3
fail_init_failure	Section 2.5.4.11.2

Table 2.2-235: turret_init Information.

2.2.6.1.1.2 turret_simul

This routine is called on a tick by tick basis to model the turret. Nothing will occur until the turret gyros are operational. When in manual mode, the stabilization is not operational. Note that the stab vectors must be set every tick, since they are set one tick ahead of use.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
old_control_value	int	Standard
Calls		
Function	**Where Described**	
turret_gyrose_simul	Section 2.2.6.1.1.12	
sound_make_const_sound	Section 2.1.3.1.2	
bcs_dump_lead_buffer	Section 2.2.3.1.1	
turret_move	Section 2.2.6.1.1.3	
make_sound_of_no_turret_noise	Section 2.2.6.1.1.35	
set_turret_vars		
turret_set_stab_sys	Section 2.5.5.2.18	
controls_turret_ref_ind	Section 2.2.2	
turret_get_ref_ind	Section 2.5.5.2.16	

Table 2.2-236: turret_simul Information.

2.2.6.1.1.3 turret_move

This routine is called by **turret_simul**() to make the turret slew and the gun elevate. It checks to make sure that the subsystems are engaged and working before the routines which actually perform the slewing and elevating are called.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
sin_stab_azi_rot	REAL	sim_types.h
sin_stab_elev_rot	REAL	sim_types.h
elev_rate	REAL	sim_types.h
slew_rate	REAL	sim_types.h
Calls		
Function	**Where Described**	
turret_get_stab_changes	Section 2.5.5.2.5	
make_sound_of_no_turret_noise	Section 2.2.6.1.1.35	
sound_of_turret_traversing	Section 2.1.3.2.10	
turret_calc_turret_slew	Section 2.2.6.1.1.7	
turret_move_azimuth	Section 2.5.5.2.6	
make_sound_of_no_elevating	Section 2.2.6.1.1.34	
turret_calc_gun_elev	Section 2.2.6.1.1.10	
turret_move_elevation	Section 2.5.5.2.7	

Table 2.2-237: turret_move Information.

2.2.6.1.1.4 turret_get_gps_slew_rate

This routine returns the gunner's primary sight slew rate. The ballistics computer system requires this information to determine lead tracking. This return value must contain only the slew rate due to the handles; it cannot include slewing due to the stab system. When lead tracking, the stab system is ignored.

Return Values		
Return Value	**Type**	**Meaning**
gps_slew_rate /DELTA_T	REAL	gps slew rate

Table 2.2-238: turret_get_gps_slew_rate Information.

2.2.6.1.1.5 turret_get_turret_slew_rate

This routine returns the turret slew rate.

Return Values		
Return Value	**Type**	**Meaning**
gps_slew_rate /DELTA_T	REAL	turret slew rate

Table 2.2-239: turret_get_gps_slew_rate Information.

2.2.6.1.1.6 turret_handles_values

This routine is called by handles to pass on the values of the gun slew rates, the gun elevation rates, which handles are engaged, and whether the fast slew is on.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
gun_slew_rate	REAL	sim_types.h
gun_elevate_rate	REAL	sim_types.h
handle_engaged	int	Standard

Table 2.2-240: turret_handles_values Information.

2.2.6.1.1.7 turret_calc_turret_slew

This routine moves the turret in azimuth. In addition to slewing the turret, this routine is also resposible for checking to make sure that the turret is not moving too fast. It also checks to see that sufficient hydraulic pressure is available before starting the turret move.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
control_handle	REAL	sim_types.h
sin_stab_azi_rot	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
total_slew_rate	REAL	sim_types.h
slew_percent	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
total_slew_rate	REAL	the total slew rate of the turret
Calls		
Function	**Where Described**	
calc_slew_from_handles	Section 2.2.6.1.1.9	
hydraulic_slew_turret_request	Section 2.2.6.4.2.10	
sound_of_turret_traversing	Section 2.1.3.2.10	
abs	sim_macros.h	

Table 2.2-241: turret_calc_turret_slew Information.

2.2.6.1.1.9 calc_slew_from_handles

This routine is called by **turret_move_azimuth**() to determine how far to slew the turret, based on the deflection of the gunner or commander's handles. The parameter, *gun_slew_handle*, is the normalized handle displacement, where -1.0 is complete deflection to the right, +1 is complete deflection to the left, and 0.0 is centered.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
handle_disp	register REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
abs_slew_handle	register REAL	sim_types.h
slew_rate	register REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
slew_rate	REAL	the slew rate
Calls		
Function	**Where Described**	
abs	sim_macros.h	
mil_to_rad	sim_macros.h	

Table 2.2-242: turret_calc_turret_slew Information.

2.2.6.1.1.10 turret_calc_gun_elev

This routine moves the gun in elevation. In addition to elevating the gun, this routine is also resposible 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.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
control_handle	REAL	sim_types.h
sin_stab_elev_rot	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
total_elev_rate	REAL	sim_types.h
hydr_elev_percent	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
total_elev_rate	REAL	the total elevation rate
Calls		
Function	**Where Described**	
calc_elev_from_handle	Section 2.2.6.1.1.11	
abs	sim_macros	
min	sim_macros	
hydraulic_elevate_gun_reque st	Section 2.2.6.4.2.11	
sound_of_gun_elevating	Section 2.1.3.2.11	

Table 2.2-243: turret_calc_gun_elev Information.

2.2.6.1.1.11 calc_elev_from_handle

This routine is called by **turret_move_elev** to determine how far to elevate the gun due to the deflection of the gunner's (or commander's) handles. The parameter, gun_elev_handle, is a number between -1.0 and +1.0, where -1.0 is a complete deflection down, +1.0 is a complete deflection up, and 0.0 is centered.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
handle_disp	register REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
abs_elev_handle	register REAL	sim_types.h
elev_rate	register REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
elev_rate	REAL	the elevation rate
Calls		
Function	**Where Described**	
abs	sim_macros.h	
mil_to_rad	sim_macros.h	

Table 2.2-245: calc_elev_from_handle Information.

2.2.6.1.1.12 turret_gyros_simul

This routine is called by **turret_simul** to simulate spinning up or spinning down of the turret gyros. The variable 'gyro_speed' is a number between 0.0 and 1.0 which represents the gyro's speed as a percentage of their full working speed.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
g_dir	register pointer to int	standard
g_speed	register pointer to REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
turret_gyros_status()	int	status of turret gyros
Calls		
Function	**Where Described**	
turret_gyros_status	Section 2.2.6.1.1.15	

Table 2.2-246: turret_gyros_simul Information.

2.2.6.1.1.13 turret_gyros_spool_up

This routine is called by controls when turret power is turned on. It spools the gyros up.

2.2.6.1.1.14 turret_gyros_spool_down

This routine is called by controls when turret power is turned off. It spools the gyros down.

2.2.6.1.1.15 turret_gyros_status

This routine is called by controls to determine the state of the turret gyros.

Return Values		
Return Value	**Type**	**Meaning**
GYROS_SPOOLED_UP	int	gyros are spooling up
GYROS_SPOOLED_DOWN	int	gyros are spooling down
GYROS_STILL_SPOOLING	int	still spooling

Table 2.2-247: turret_gyros_status Information.

2.2.6.1.1.16 turret_break_gearbox

This routine breaks the gearbox by seting gearbox_status to BROKEN.

2.2.6.1.1.17 turret_repair_gearbox

This routine repairs the gearbox by setting gearbox_status to WORKING.

2.2.6.1.1.18 turret_break_elevation_drive

This routine breaks the elevation drive by setting elevation_status to BROKEN.

2.2.6.1.1.19 turret_repair_elevation_drive

This routine repairs the elevation drive by setting elevation_status to WORKING.

2.2.6.1.1.20 turret_break_stab_system

This routine breaks the stabalization system by setting stab_status to BROKEN.

2.2.6.1.1.21 turret_repair_stab_system

This routine repairs the stabalization system by setting stab_status to WORKING.

2.2.6.1.1.22 turret_break_mount_interface

This routine causes the mount interface to fail.

2.2.6.1.1.23 turret_repair_mount_interface

This routine repairs the mount interface.

2.2.6.1.1.24 turret_break_traverse_drive

This routine causes the traverse drive to fail.

2.2.6.1.1.25 turret_repair_traverse_drive

This routine repairs the traverse drive.

2.2.6.1.1.26 turret_fire_control_emergency

This routine sets the fire control mode to FCM_EMERGENCY.

2.2.6.1.1.27 turret_fire_manual

This routine sets the fire control mode to FCM_MANUAL.

2.2.6.1.1.28 turret_fire_control_normal

This routine sets the fire control mode to FCM_NORMAL.

2.2.6.1.1.29 turret_gun_turret_drive_uncoupled

This routine uncouples the gun and the turret.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-248: turret_gun_turret_drive_uncoupled Information.

2.2.6.1.1.30 turret_gun_turret_drive_powered

This routine powers the gun turret drive.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-249: turret_gun_turret_drive_powered Information.

2.2.6.1.1.31 turret_gun_turret_drive_manual

This routine sets the gun turret drive to manual.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-250: turret_gun_turret_drive_manual Information.

2.2.6.1.1.32 turret_collision_detected

This routine is called by kinematics whenever a collision is detected. It determines whether the gun was pointing in the direction of the collision. If so, it checks to see whether to break the turret-mount interface. When the turret-mount interface is broken, the gun cannot be elevated (except in emergency mode) or fired.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
agent_id	pointer to VehicleId	basic.h
event_id	int	Standard
coll_sector	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
azimuth	register REAL	sim_types.h
rel_sector	register int	Standard
t_to_h	register T_MATRIX	sim_types.h
Calls		
Function	**Where Described**	
rad_to_mil	sim_macros.h	
tracks_compute_velocity	Section 2.2.6.2.3.10	
fail_break_system	Section 2.5.4.8.1	

Table 2.2-251: calc_elev_from_handle Information.

2.2.6.1.1.33 make_sound_of_no_slewing

This routine causes the sound of the turret traversing without slewing to be made.

Calls	
Function	Where Described
sound_of_turret_traversing	Section 2.1.3.2.10

Table 2.2-252: make_sound_of_no_slewing Information.

2.2.6.1.1.34 make_sound_of_no_elevating

This routine causes the sound of the turret moving without elevating to be made.

Calls	
Function	Where Described
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-253: make_sound_of_no_elevating Information.

2.2.6.1.1.35 make_sound_of_no_turret_noise

This routine suppresses the sound of turret motion.

Calls	
Function	Where Described
make_sound_of_no_slewing	Section 2.2.6.1.1.33
make_sound_of_no_elevatin g	Section 2.2.6.1.1.34

Table 2.2-254: turret_gun_turret_drive_manual Information.

2.2.6.1.1.36 turret_get_gun_to_world

This routine calculates the gun to world transformation matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
g_to_w	T_MATRIX	sim_types.h
error	VECTOR	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
lead_azimuth	REAL	sim_types.h
Calls		
Function	**Where Described**	
bcs_set_ballistics_computer	Section 2.2.3.1.13	
bcs_get_lead_azimuth	Section 2.2.3.1.14	
bcs_get_superelevation	Section 2.2.3.1.15	
turret_get_g_to_w	Section 2.5.5.2.13	

Table 2.2-255: turret_get_gun_to_world Information.

2.2.6.1.2 m1_cupola.c

(./simnet/release/src/vehicle/m1/src/m1_cupola.c [m1_cupola.c])

The periscope views for the commander in the hatch are modeled by a rotating cupola with attached viewports. The commander turns the cupola by pressing a switch. The loader in the M1 has a periscope as well. He must physically turn the periscope just as he does in a real tank. In both cases, the controls code determines the position of the cupola as a percentage of its full range. M1_cupola.c determines the angle of the cupola with respect to the turret. The sine and cosine are made available to the CIG so the appropriate image can be drawn in the periscope viewports.

Includes:

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

Defines:

CWS_FIELD_OF_VIEW -- 300 degrees
LPSCOPE_FIELD_OF_VIEW -- 330 degrees
SPSCOPE_MOUNT_OFFSET -- 77 degrees

Declarations:
cws_sin = 0.0
cws_cos = 1.0
lpscope_sin = 0.0
lpscope_cos = 1.0
new_cws_value = TRUE
new_lpscope_value = TRUE
cws_current_offset = 0.0
lpscope_current_offset = 0.0

2.2.6.1.2.1 convert_disp_to_angle

This routine sets the values pointed to by *psin* and *pcos* to the sine and cosine of the angle calculated from the displacement and offset arguments (*disp* and *offset*)

Parameters		
Parameter	**Type**	**Where Typedef Declared**
disp	REAL	sim_types.h
fov	REAL	sim_types.h
psin	pointer to REAL	sim_types.h
pcos	pointer to REAL	sim_types.h
offset	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
angle	register REAL	sim_types.h

Table 2.2-256: convert_disp_to_angle Information.

2.2.6.1.2.2 cupola_cws_new_value

This routine sets the value of the commander weapon system offset to the value passed in *val.*

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h

Table 2.2-257: cupola_cws_new_value Information.

2.2.6.1.2.3 cupola_lscope_new_value

This routine sets the value of the loader's periscope offset to the value passed in *val.*

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h

Table 2.2-258: cupola_lscope_new_value Information.

2.2.6.1.2.4 cupola_simul

This routine performs the tick by tick simulation of the cupola. The routine determines whether the cupola moved, and if so, calculates the new commander weapon system or loader's periscope sine and cosine.

Calls	
Function	**Where Described**
convert_disp_to_angle	Section 2.2.6.1.2.1

Table 2.2-259: cupola_simul Information.

2.2.6.1.2.5 cupola_init

This routine initializes the value of the commander weapon system and loader's periscope offsets at 0.0.

2.2.6.2 Propulsion Simulation

The following CSU's model the M1's propulsion :

m1_dtrain.c
m1_engine.c
m1_tracks.c

2.2.6.2.1 m1_dtrain.c
(./simnet/release/src/vehicle/m1/src/m1_dtrain.c [m1_dtrain.c])

This module Simulates the Detroit Diesel Allison X11000-3B hydrokinetic automatic transmission and torque converter, final drive, and drive sprocket. Failure modes for the transmission, oil temperature and pressure are also simulated.

The following are included:

"stdio.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"libfail.h"
"failure.h"
"libsound.h"
"m1_cntrl.h"
"m1_dtrain.h"
"m1_engine.h"
"m1_sound_dfn.h"
"m1_tracks.h"

The following are defined:

TORQUE_CONVERTER_SLOPE	
TORQUE_CONVERTER_INTERCEPT	
TORQUE_CONVERTER_VELOCITY_LOAD_COEFFICIENT	
R	Tranmission lever settings
D	
L	
N	
PVT	
NEUTRAL	Gears
PIVOT_STEER	
F1	
F2	
F3	
F4	
R1	
R2	
NEUTRAL_RATIO	Gear Ratios
PIVOT_STEER_RATIO	
F1_RATIO	
F2_RATIO	
F3_RATIO	
F4_RATIO	

R1_RATIO	
R2_RATIO	
FINAL_DRIVE_RATIO	
F1_INERTIA	Gear inertias(ft-lb-sec^2)
F2_INERTIA	
F3_INERTIA	
F4_INERTIA	
R1_INERTIA	
R2_INERTIA	
NEUTRAL_INERTIA	
PIVOT_STEER_INERTIA	
DRIVETRAIN_MISC_INERTIA	
SLIP_INERTIA	
TC1	Torque Converter Multiplier Constants
TC2	
SPROCKET_RADIUS	ft
TANK_WEIGHT	lbs
TRACKS_WEIGHT	lbs
CLUTCH_LOCKUP_SPEED	rpm
UPSHIFT_SPEED	rpm
DOWNSHIFT_SPEED	rpm
REV_DOWNSHIFT_SPEED	rpm
MIN_TRANSMISSION_OIL_PUMP_SPEED	rpm
MIN_TC_STEER_SPEED	rpm
MAX_TC_STEER_SPEED	rpm
MAX_PIVOT_FD_SPEED	
SPROCKET_MPH_TO_RPM_CONVERSION	rpm
TWOPI	
MAX_STEERING_SPROCKET_SPEED_INCREMENT	rpm (48.84 / 2.0)
MAXIMUM_SERVICE_BRAKE_TORQUE	ft-lbs
MAXIMUM_PARKING_BRAKE_TORQUE	ft-lbs
BRAKING_STICTION_FACTOR	
SERVICE_BRAKE_LOCKUP_INCREMENT	
LOW	
NORMAL	
HIGH	

The following variables are declared:

torque_converter_speed	TC output rpm = gearbox input rpm
torque_converter_torque	TC output torque=gearbox input torque
gearbox_speed	gearbox output rpm = FD input rpm
sprocket_grav_torque	load torque due to gravity
sprocket_drag_torque	load torque due to drag
left_sprocket_speed	RPM (adjusted for steering)
right_sprocket_speed	RPM (adjusted for steering)
net_drive_torque	
final_drive_torque	output torque
final_drive_grav_torque	load torque on FD due to gravity
final_drive_drag_torque	load torque on FD due to drag
final_drive_brake_torque	load torque on FD due to braking
final_drive_speed	FD output rpm = sprocket rpm
gearbox_torque	
steering_bar	Steering bar position (-1.0 to 1.0)
slip_state	

gear	NEUTRAL,F1,F2,F3,F4,R1,R2
gear_ratio;	
transmission_select	R, D, N, L , PVT
transmission_oil_temperature	NORMAL or HIGH
transmission_oil_pressure	NORMAL or LOW
transmission_oil_level	NORMAL or LOW
service_brake	
parking_brake	
service_brake_status	
parking_brake_status	
transmission_failure_status	
transmission_oil_filter_status	

Folowing are the Lumped Moments of Inertia (ft-lb-sec^2) for the entire drivetrain and transformed vehicle weight as they appear at the torque converter after undergoing the appropriate gear reductions. Note: These are variables which are computed once at initialization and used as constants during the course of the simulation. The variable names are all capitalized (against convention):

F1_LUMPED_INERTIA
F2_LUMPED_INERTIA
F3_LUMPED_INERTIA
F4_LUMPED_INERTIA
R1_LUMPED_INERTIA
R2_LUMPED_INERTIA
NEUTRAL_LUMPED_INERTIA
PIVOT_STEER_LUMPED_INERTIA
TANK_WEIGHT_INERTIA
SPROCKET_MPH_TO_RPM_CONVERSION
RADSEC_TO_RPM_CONVERSION

2.2.6.2.1.1 drivetrain_load_torque_converter

This routine computes the load torque on the torque converter given the engine speed, *rpm*..

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rpm	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
load_torque	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
load_torque	REAL	the load torque

Table 2.2-260: drivetrain_load_torque_converter Information.

2.2.6.2.1.2 drivetrain_lockup_clutch

This routine determines if the torque converter is locked up.

Return Values		
Return Value	**Type**	**Meaning**
ON	int	clutch locked up
OFF	int	not locked up

Table 2.2-261: drivetrain_lockup_clutch Information.

2.2.6.2.1.3 drivetrain_torque_converter_speed

This routine returns the torque converter output speed in rpm.

Return Values		
Return Value	**Type**	**Meaning**
torque_converter_speed	REAL	torque converter speed

Table 2.2-262: drivetrain_torque_converter_speed Information.

2.2.6.2.1.4 drivetrain_neutral

This routine sets the transmission direction selection to neutral.

2.2.6.2.1.5 drivetrain_low

This routine sets the transmission direction selection to low.

2.2.6.2.1.6 drivetrain_drive

This routine sets the transmission direction selection to drive.

2.2.6.2.1.7 drivetrain_reverse

This routine sets the transmission direction selection to reverse.

2.2.6.2.1.8 drivetrain_pivot

This routine sets the transmission direction selection to pivot.

2.2.6.2.1.9 drivetrain_set_steering_bar

This routine sets the steering bar position to the passed value *val.*

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h

Table 2.2-263: drivetrain_set_steering_bar Information.

2.2.6.2.1.10 drivetrain_set_service_brake

This routine sets the service brake position to the passed value *val.*

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h
Calls		
Function	**Where Described**	
sound_make_const_sound	Section 2.1.3.1.2	

Table 2.2-264: drivetrain_set_service_brake Information.

2.2.6.2.1.11 drivetrain_set_parking_brake

This routine sets the parking brake, making the appropriate sound effect.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-265: drivetrain_set_parking_brake Information.

2.2.6.2.1.12 drivetrain_release_parking_brake

This routine releases the parking brake, making the appropriate sound effect.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-266: drivetrain_release_parking_brake Information.

2.2.6.2.1.13 drivetrain_service_brake_failure

This routine causes the failure of the service brake.

2.2.6.2.1.14 drivetrain_parking_brake_failure

This routine causes the failure of the parking brake.

2.2.6.2.1.15 drivetrain_repair_service_brake

This routine repairs the service brake.

2.2.6.2.1.16 drivetrain_repair_parking_brake

This routine repairs the parking brake.

2.2.6.2.1.17 drivetrain_transmission_select_neutral

This routine returns TRUE if the drivetrain is in neutral and returns FALSE otherwise.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	in neutral
FALSE	int	not in neutral

Table 2.2-267: drivetrain_transmission_select_neutral Information.

2.2.6.2.1.18 load_sprocket

This routine determines the load on the tracks sprocket.

Calls	
Function	**Where Described**
tracks_return_slip_state	Section 2.2.6.2.3.28
tracks_compute_friction_force	Section 2.2.6.2.3.1
tracks_compute_gravity_load	Section 2.2.6.2.3.3
tracks_compute_drag_load	Section 2.2.6.2.3.4
tracks_compute_velocity	Section 2.2.6.2.3.10

Table 2.2-268: load_sprocket Information.

2.2.6.2.1.19 compute_fd_brake_torque

This routine returns the total brake torque.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
service_brake_torque	REAL	sim_types.h
parking_brake_torque	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
parking_brake_torque + service_brake_torque	REAL	total brake torque

Table 2.2-269: compute_fd_brake_torque Information.

2.2.6.2.1.20 get_braking_factor

This routine calculates the braking factor.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
braking_factor	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
braking_factor	REAL	the braking factor
service_brake	REAL	service braking factor

Table 2.2-270: get_braking_factor Information.

2.2.6.2.1.21 load_final_drive

This routine determines the final drive load.

Calls	
Function	**Where Described**
compute_fd_brake_torque	Section 2.2.6.2.1.19

Table 2.2-271: load_final_drive Information.

2.2.6.2.1.22 set_gear_ratio

This routine sets the gear ratio.

2.2.6.2.1.23 gearbox_shift

This routine controls the gearbox shift.

Calls	
Function	**Where Described**
fail_break_system	Section 2.5.4.8.1
set_gear_ratio	Section 2.2.6.2.1.22

Table 2.2-272: gearbox_shift Information.

2.2.6.2.1.24 load_gearbox

This routine calculates the torque converter speed and shifts the gearbox if necessary.

Calls	
Function	**Where Described**
gearbox_shift	Section 2.2.6.2.1.23

Table 2.2-273: load_gearbox Information.

2.2.6.2.1.25 power_gearbox

This routine calculates the gearbox torque.

2.2.6.2.1.26 current_fd_inertia

This routine calculates the final drive inertia.

Return Values		
Return Value	**Type**	**Meaning**
inertia	REAL	final drive inertia

Table 2.2-274: current_fd_inertia Information.

2.2.6.2.1.27 power_final_drive

This routine powers the final drive. It contains the integrator for the entire drivetrain.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
new_fd_speed	REAL	sim_types.h
Calls		
Function	**Where Described**	
current_fd_inertia	Section 2.2.6.2.1.26	

Table 2.2-275: current_fd_inertia Information.

2.2.6.2.1.28 differential steer

This routine calculates the left and right sprocket speeds with the steering differential.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
differential_steer_fd_rpm	REAL	sim_types.h
tc_speed_ratio	REAL	sim_types.h
torque_converter_input_speed	REAL	sim_types.h
Errors		
Error Name	**Reason for Error**	
Steer Error	Unknown gear	
Calls		
Function	**Where Described**	
engine_get_speed	Section 2.2.6.2.2.11	
get_braking_factor	Section 2.2.6.2.1.20	

Table 2.2-276: differential_steer Information.

2.2.6.2.1.29 power_sprocket

This routine calculates the left and right track speeds in miles per hour and sends the torque and speed to the tracks.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
left_speed	REAL	sim_types.h
right_speed	REAL	sim_types.h
torque	REAL	sim_types.h

Calls	
Function	**Where Described**
differiential_steer	Section 2.2.6.2.1.28
tracks_send_velocities	Section 2.2.6.2.3.14

Table 2.2-277: power_sprocket Information.

2.2.6.2.1.30 power_engine

This routine performs the engine simulation.

Calls	
Function	**Where Described**
engine_simul	Section 2.2.6.2.2.7

Table 2.2-278: power_engine Information.

2.2.6.2.1.31 compute_lumped_inertias

This routine computes the equivalent lumped inertia of all elements in the drivetrain, including the effect of the weight of the tank as it appears to the final drive for the various gear positions. Note: this routine is only called once, during initialization.

2.2.6.2.1.32 compute_compile_time_constants

This routine is called once at initialization to compute compile time constants.

Calls	
Function	**Where Described**
compute_lumped_inertias	Section 2.2.6.2.1.31

Table 2.2-279: compute_compile_time_constants Information.

2.2.6.2.1.33 power_torque_converter

This routine computes the torque converter torque.

Calls	
Function	**Where Described**
engine_get_torque	Section 2.2.6.2.2.11
engine_get_speed	Section 2.2.6.2.2.12

Table 2.2-280: power_torque_converter Information.

2.2.6.2.1.34 load_drivetrain

This routine calculates the drivetrain load by calling the sprocket, final drive, and gearbox load functions.

Calls	
Function	**Where Described**
load_sprocket	Section 2.2.6.2.1.18
load_final_drive	Section 2.2.6.2.1.21
load_gearbox	Section 2.2.6.2.1.24

Table 2.2-281: load_drivetrain Information.

2.2.6.2.1.35 power_drivetrain

This routine calculates power on each drivetrain component.

Calls	
Function	**Where Described**
power_engine	Section 2.2.6.2.1.30
power_torque_converter	Section 2.2.6.2.1.33
power_gearbox	Section 2.2.6.2.1.25
power_final_drive	Section 2.2.6.2.1.27
power_sprocket	Section 2.2.6.2.1.29

Table 2.2-282: power_drivetrain Information.

2.2.6.2.1.36 send_transmission_oil_status

This routine sends status information to the transmission oil temperature indicators and oil pressure indicators.

Calls	
Function	**Where Described**
controls_transmission_oil_temperature_high	Section 2.2.2
controls_transmission_oil_temperature_normal	Section 2.2.2
controls_transmission_oil_pressure_low	Section 2.2.2
controls_transmission_oil_pressure_normal	Section 2.2.2

Table 2.2-283: send_transmission_oil_status Information.

2.2.6.2.1.37 send_trans_maintenance_status

This routine sends status infomration to the transmission oil level and oil filter status indicators.

Calls	
Function	**Where Described**
controls_transmission_oil_level_low	Section 2.2.2
controls_transmission_oil_filter_clogged	Section 2.2.2
controls_transmission_oil_level_normal	Section 2.2.2
controls_transmission_oil_filter_normal	Section 2.2.2

Table 2.2-284: send_trans_maintenance_status Information.

2.2.6.2.1.38 send_dtrain_outputs

This routine sends the drivetrain output status to the indicators in the controls module.

Calls	
Function	**Where Described**
send_transmission_oil_status	Section 2.2.6.2.1.36
send_trans_maintenance_status	Section 2.2.6.2.1.37

Table 2.2-285: send_dtrain_outputs Information.

2.2.6.2.1.39 transmission_oil_system_simul

This routine performs the transmission oil system simulation. It calculates the status of oil temperature, oil pressure, oil filter status, and oil level.

Calls	
Function	**Where Described**
engine_get_speed	Section 2.2.6.2.2.11

Table 2.2-286: transmission_oil_system_simul Information.

2.2.6.2.1.40 drivetrain_simul

This routine performs the drivetrain simulation. It is called on a tick by tick basis. It calculates the drive train load and performs the oil system simulation and tracks simulation. It sends output information to the controls system.

Calls	
Function	**Where Described**
load_drivetrain	Section 2.2.6.2.1.34
power_drivetrain	Section 2.2.6.2.1.35
transmission_oil_system_simul	Section 2.2.6.2.1.39
tracks_simul	Section 2.2.6.2.3.25
send_dtrain_outputs	Section 2.2.6.2.1.38

Table 2.2-287: drivetrain_simul Information.

2.2.6.2.1.41 drivetrain_init

This routine initializes the drivetrain. It initializes the status, sets initial settings, places the tank in neutral, sets the parking brake and initializes the failures package.

Calls	
Function	**Where Described**
engine_init	Section 2.2.6.2.2.22
tracks_init	Section 2.2.6.2.3.16
compute_compile_time_comstants	Section 2.2.6.2.1.32
controls_set_parking_brake	Section 2.2.2
fail_init_failure	Section 2.5.2.5.4.11.2

Table 2.2-288: drivetrain_init Information.

2.2.6.2.1.42 drivetrain_clog_transmission_oil_filter

This routine clogs the transmission oil filter.

2.2.6.2.1.43 drivetrain_replace_transmission_oil_filter

This routine replaces the transmission oil filter.

2.2.6.2.1.44 drivetrain_transmission_oil_leak

This routine causes a transmission oil leak.

2.2.6.2.1.45 drivetrain_repair_transmission_oil_leak

This routine repairs a transmission oil leak.

2.2.6.2.1.46 drivetrain_refill_transmission_oil

This routine is not used in the Version 6.6 release.

2.2.6.2.1.47 drivetrain_replace_transmission

This routine restores the transmission.

2.2.6.2.1.48 drivetrain_transmission_failure

This routine causes a transmission failure.

Calls	
Function	**Where Described**
drivetrain_transmission_oil_leak	Section 2.2.6.2.1.44

Table 2.2-290: drivetrain_transmission_failure Information.

2.2.6.2.2 m1_engine.c

(./simnet/release/src/vehicle/m1/src/m1_engine.c [m1_engine.c])

The AVCO Lycoming AGT-1500 gas turbine engine is simulated by obtaining the desired power setting from the throttle input, and computing the torque output from a linear torque curve model. The engine maintains its own dynamic state based on engine inertia, transmission load, and torque output. Failures are maintained for oil pressure, oil temperature, oil filter, coolant temperature, and fuel filter. The simulation routines are accessed from m1_dtrain.c, and the failure routines from m1_failure.c.These routines are contained in the CSU m1_engine.c.

The following are included:

"stdio.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"dynlib.h"
"libfail.h"
"failure.h"
"libsound.h"
"m1_engine.h"
"m1_dtrain.h"
"m1_sound_dfn.h"
"m1_cntrl.h"
"m1_meter.h"

The following are defined:

ENGINE_TORQUE_INTERCEPT	
ENGINE_TORQUE_SLOPE	
ENGINE_INERTIA	in ft-lbs-sec^2
TACTICAL_IDLE_SPEED	rpm
LOW_IDLE_SPEED	rpm
MIN_ENGINE_OIL_PUMP_SPEED	rpm
GOVERNOR_DROOP_SPEED	rpm
MAX_RATED_ENGINE_SPEED	rpm
ENGINE_OVERSPEED_RPM	rpm
MAX_RUNAWAY_ENGINE_SPEED	rpm
KGOV_DROOP	Governor power droop proportional to gain
TWOPI	
IDLE_POWER_SETTING 0.06	Produces slow creep (~2 mph) in F2
ENGINE_POWER_TIME_CONSTANT	sec
ENGINE_IDLE_TIME_CONSTANT	
SPECIFIC_FUEL_CONSUMPTION	gal / HP-hr
RPM_FT_LBS_TO_HP_CONVERSION	twopi / 33000
CLOGGED_FILTER_POWER_LOSS	
MAX_FILTER_CLOGGED_TIME	sec (= 30 min)
MAX_OIL_LEVEL_LOW_TIME	sec (= 30 min)
SPOOLING_UP	
SPOOLING_DOWN	
RUNAWAY	if master power off
SPOOLING_TIME	spool up/down time (sec)
LOW	

NORMAL
HIGH

The following variables are declared:

engine_speed;	rpm
engine_torque;	ft-lbs
throttle;	0 - 1.0
power_setting;	0 - 1.0
engine_max_available_power;	0 - 1.0
tac_idle;	ON or OFF
engine_fuel_flow;	gal/hr
engine_failure_status;	WORKING or BROKEN
engine_starter_status;	WORKING or BROKEN
engine_pilot_relay_status;	WORKING or BROKEN
engine_cooling_system_status;	WORKING or BROKEN
engine_oil_filter_status;	WORKING or BROKEN (BROKEN = CLOGGED)
engine_fuel_filter_status;	WORKING or BROKEN (BROKEN = CLOGGED)
engine_status;	ON, OFF, SPOOLING_UP, SPOOLING_DOWN
engine_oil_pressure;	NORMAL or LOW
engine_oil_temperature;	NORMAL or HIGH
engine_oil_level;	NORMAL or LOW
engine_oil_level_low_timer;	sec
engine_oil_filter_clogged_timer	sec
SPOOLING_INCREMENT	computed once at init
RUNAWAY_SPOOLING_INCREMENT	computed once at init

2.2.6.2.2.1 set_power

This routine calculates the throttle power setting.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
target_power_setting	REAL	sim_types.h
throttle_power_setting	REAL	sim_types.h

Calls	
Function	**Where Described**
first_order_lag	Section 2.5.7.6.1
max	sim_macros.h (macro definition)

Table 2.2-291: set_power Information.

2.2.6.2.2.2 compute_fuel_consumption

This routine computes the fuel consumption based on engine speed and torque.

Calls	
Function	**Where Described**
fuel_set_flow	Section 2.2.5.2.8
spool_down_engine	Section 2.2.6.2.2.19

Table 2.2-292: compute_fuel_consumption Information.

2.2.6.2.2.3 engine_dynamics

This routine simulates the engine dynamics. It calculates engine speed, torque converter load and speed, and calls the routine to calculate fuel consumption.

Calls	
Function	**Where Described**
set_power	Section 2.2.6.2.2.1
drivetrain_lockup_clutch	Section 2.2.6.2.1.2
first_order_lag	Section 2.5.7.6.1
drivetrain_load_torque_converter	Section 2.2.6.2.1.1
drivetrain_torque_converter_speed	Section 2.2.6.2.1.3
compute_fuel_consumption	Section 2.2.6.2.2.2

Table 2.2-293: engine_dynamics Information.

2.2.6.2.2.4 send_engine_sound

This routine determines the engine sound to make based on the engine status.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
engine_sound	REWAL	sim_types.h
Calls		
Function	**Where Described**	
sound_make_var_sound	Section 2.1.3.1.4	

Table 2.2-294: send_engine_sound Information.

2.2.6.2.2.5 send_engine_controls_status

This routine sends the status of the engine to controls in order to set the instrument readings.

Calls	
Function	**Where Described**
controls_engine_oil_pressure_low	Section 2.2.2
controls_engine_oil_pressure_normal	Section 2.2.2
controls_engine_oil_temperature_high	Section 2.2.2
controls_engine_oil_temperature_normal	Section 2.2.2
controls_engine_oil_level_low	Section 2.2.2
controls_engine_oil_level_nomal	Section 2.2.2
controls_engine_oil_filter_clogged	Section 2.2.2
controls_engine_oil_filter_normal	Section 2.2.2
controls_engine_fuel_filter_clogged	Section 2.2.2
controls-engine_fuel_filter-normal	Section 2.2.2
controls_engine_overspeed	Section 2.2.2
controls_engine_overspeed_normal	Section 2.2.2

Table 2.2-295: send_engine_controls_status Information.

2.2.6.2.2.6 send_all_outputs

This routine sends outputs to the various output devices: controls status to the controls, tachometer settings to the indicators, and sounds to the sound system.

Calls	
Function	**Where Described**
send_engine_sound	Section 2.2.6.2.2.4
meter_tach_set	Section 2.2.2
send_engine_controls_status	Section 2.2.6.2.2.5

Table 2.2-296: send_all_outputs Information.

2.2.6.2.2.7 engine_oil_system_simul

This routine simulates the engine oil system: oil pressure, oil level, oil temperature, oil filter, and failures.

Calls	
Function	**Where Described**
fail_break_system	Section 2.5.4.8.1

Table 2.2-297: engine_oil_system_simul Information.

2.2.6.2.2.7 engine_simul

This routine simulates the engine system: the dynamics, engine speed, status, failures, and outputs.

Calls	
Function	**Where Described**
engine_dynamics	Section 2.2.6.2.2.3
controls_engine_started	Section 2.2.2
fail_break_system	Section 2.5.4.8.1
spool_down_engine	Section 2.2.6.2.2.19
controls_engine_abort	Section 2.2.2
engine_oil_system_simul	Section 2.2.6.2.2.7
send_all_outputs	Section 2.2.6.2.2.6

Table 2.2-298: engine_simul Information.

2.2.6.2.2.8 engine_running

This routine returns TRUE if the engine is running and returns FALSE otherwise.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	engine running
FALSE	int	not running

Table 2.2-299: engine_running Information.

2.2.6.2.2.9 engine_spooling_up

This routine returns TRUE if the engine is spooling up and returns FALSE otherwise.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	engine spooling up
FALSE	int	not running

Table 2.2-300: engine_spooling_up Information.

2.2.6.2.2.10 engine_spooling_down

This routine returns TRUE if the engine is spooling down and returns FALSE otherwise.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	engine spooling down
FALSE	int	not running

Table 2.2-301: engine_spooling-down Information.

2.2.6.2.2.11 engine_get_speed

This routine returns the engine speed.

Return Values		
Return Value	**Type**	**Meaning**
engine_speed	REAL	engine speed

Table 2.2-302: engine_get_speed Information.

2.2.6.2.2.12 engine_get_torque

This routine returns the engine torque.

Return Values		
Return Value	**Type**	**Meaning**
engine_torque	REAL	engine torque

Table 2.2-303: engine_get_torque Information.

2.2.6.2.2.13 engine_get_power

This routine returns the engine power.

Return Values		
Return Value	**Type**	**Meaning**
engine_power	REAL	engine power

Table 2.2-304: engine_get_power Information.

2.2.6.2.2.14 engine_get_max_power

This routine returns the maximum available power of the engine.

Return Values		
Return Value	**Type**	**Meaning**
engine_max_available_power	REAL	maximum available power

Table 2.2-305: engine_get_max_power Information.

2.2.6.2.2.15 engine_tac_idle_switch_on

This routine sets *tac_idle* to ON.

2.2.6.2.2.16 engine_tac_idle_switch_on

This routine sets *tac_idle* to OFF.

2.2.6.2.2.17 engine_set_throttle

This routine sets the value of the throttle.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL`	sim_types.h

Table 2.2-306: engine_set_throttle Information.

2.2.6.2.2.18 engine_start_switch

This routine starts the engine.

Calls	
Function	**Where Described**
drivetrain_transmission_select_neutral	Section 2.2.6.2.1.17
sound_make_const_sound	Section 2.1.3.1.2
electsys_engine_start_request	Section 2.2.6.3.1.8
controls_engine_abort	Section 2.2.2

Table 2.2-307: engine_start_switch Information.

2.2.6.2.2.19 spool_down_engine

This routine spools down the engine.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2
controls_engine_spooling_down	Section 2.2.2

Table 2.2-308: spool_down_engine Information.

2.2.6.2.2.20 engine_shutoff_switch

This routine spools down the engine and shuts it off when the shutoff switch is selected.

Calls	
Function	**Where Described**
spool_down_engine	Section 2.2.6.2.2.19
controls_engine_abort	Section 2.2.2

Table 2.2-309: engine_shutoff_switch Information.

2.2.6.2.2.21 compute_engine_compile_time_constants

This routine computes the spooling increment and runaway spooling increment at compile time.

2.2.6.2.2.22 engine_init

This routine initializes the engine and sets the status of all of the sybsystems.

Calls	
Function	**Where Described**
compute_engine_compile_time_constants	Section 2.2.6.2.2.21
fail_init_failure	Section 2.5.4.11.2

Table 2.2-310: engine_init Information.

2.2.6.2.2.23 engine_major_failure

This routine breaks the engine as a system.

2.2.6.2.2.24 engine_replace_powerpack

This routine repairs the power pack.

Calls	
Function	**Where Described**
drivetrain_replace_transmission	Section 2.2.6.2.1.47

Table 2.2-311: engine_replace_powerpack Information.

2.2.6.2.2.25 engine_runaway_condition

This routine sets the runaway condition.

2.2.6.2.2.26 engine_fix_runaway_condition

This routine repairs the runaway condition.

2.2.6.2.2.27 starter_failure

This routine causes the engine starter to fail.

2.2.6.2.2.28 engine_replace_starter

This routine replaces the engine starter.

2.2.6.2.2.29 engine_pilot_relay_failure

This routine causes the pilot relay to fail.

2.2.6.2.2.30 engine_replace_pilot_relay

This routine replaces the pilot relay.

2.2.6.2.2.31 engine_clog_oil_filter

This routine causes the oil filter to be clogged.

2.2.6.2.2.32 engine_replace_oil_filter

This routine replaces the oil filter.

2.2.6.2.2.33 engine_oil_leak

This routine causes an engine oil leak.

2.2.6.2.2.34 engine_degrade_engine_power

This routine causes a degradation of engine power to a maximum passed value, *value*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
value	REAL	sim_types.h

Table 2.2-312: engine_degrade_engine_power Information.

2.2.6.2.2.35 engine_refill_oil

This routine sets the engine oil level to normal.

2.2.6.2.2.36 engine_cooling_system_failure

This routine causes the engine cooling system to fail.

2.2.6.2.2.37 engine_repair_cooling_system

This routine repairs the engine's cooling system.

2.2.6.2.2.38 engine_clog_fuel_filter

This routine causes the fuel filter to be clogged and degrades the engine power proportionally.

2.2.6.2.2.39 engine_replace_fuel_filter

This routine replaces the fuel filter and causes the available power to be increased.

2.2.6.2.3 m1_tracks.c

(./simnet/release/src/vehicle/m1/src/m1_tracks.c [m1_tracks.c])

The tracks and hull dynamics are simulated in this module. Torque from the differential is sent to both tracks which accelerate and rotate the tank. Frictional forces are computed for traction behavior. Failure modes simulated are thrown tracks. The simulation routines are called from m1_dtrain.c, and the failure routines from m1_failure.c. These routines are contained within the CSU m1_tracks.c.

This file consists of the M1 track simulation module. It includes the terrain slippage and interaction model. It also includes the bigwheel algorithm to compute the vehicle's support plane.

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_data.h"
"libterrain.h"
"libkin.h"
"libsusp.h"
"libfail.h"
"failure.h"
"libhull.h"
"bigwheel.h"
"veh_appear.h"
"libfail.h"
"libsound.h"
"m1_dtrain.h"
"m1_meter.h"
"m1_repair.h"
"m1_sound.h"
"m1_sound_dfn.h"
"m1_tracks.h"

Defines:

TRACKS_DEBUG	
TANK_WEIGHT	lbs
FWD_DRAG_BUILDUP_SPEED	mph
REV_DRAG_BUILDUP_SPEED	
DRAG_BUILDUP_GAIN	
TRACK_SEPARATION_DISTANCE	meters (= 112.0 in)
MAX_TRACK_CANT_SIN	sin (22 degrees) = 40%
MAX_RCI050_TRACK_SPEED_DIFFERENTIAL	mph
RCI_250_CONSTANT_DRAG_FORCE	lbs
RCI_250_DRAG_SPEED_COEFFICIENT	
RCI_050_CONSTANT_DRAG_FORCE	lbs
RCI_050_DRAG_SPEED_COEFFICIENT	

MUCK_DRAG_FORCE	
THROWN_TRACK_DRAG_FORCE	
TRACK_SELF_REPAIR_TIME	minutes
MAX_RATED_TRACK_SPEED	(mph) For purposes of sound only
TRACK_THROWN_OFFSET	meters
WALL_CLIMBING_HEIGHT	meters
SPROCKET_RADIUS	feet
LEVER_ARM	Meters
ANGLE_LIM	~9 degrees .7 m by 4.5 m
GUN_FORCE	Force of firing the gun

Declarations:

mean_track_velocity	mph
left_track_velocity	mph
right_track_velocity	mph
vehicle_actual_velocity	mph
old_vehicle_actual_velocity	mph
vehicle_acceleration	mph
vehicle_actual_turn_rate	rad/sec
left_track_status	WORKING or BROKEN
right_track_status	WORKING or BROKEN
soil_type	RCI-250 or RCI-50 or WATER
skid_sound_counter	for skidding sound
odometer_reading	elapsed km
odometer_count	tenths of km
elapsed_mileage	miles (float)
mileage_count	miles (int)
slip_state	NO_SLIP or SLIPPING
ground_force	force exerted on ground by treads
coefficient_of_friction[2][6]	2 states, 6 soil types
old_track_velocity	for slip calc
increment	velocity increment per tick

These parameters are used to run-time initialize the bigwheel library.

rear_wheel	
left_wheel	
right_wheel	
TRACK_THROWN_OFFSET	meters
WALL_CLIMBING_HEIGHT	meters

These parameters are used to run-time initialize the suspension library.

ROT_WN	rot suspension natural freq (rad)
ROT_ZETA	rotational suspension damping ratio
SIDE_WN	side suspension natural freq (rad)
SIDE_ZETA	side suspension damping ratio

2.2.6.2.3.1 tracks_compute_friction_factor

This routine calculates the tracks friction factor.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
force	REAL	sim_types.h
pitch_sin	REAL	sim_types.h
pitch_cos	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
force	REAL	friction force
Calls		
Function	**Where Described**	
kinematics_pitch_sin	Section 2.5.8.5.1	
sqrt	sim_macros.h	
square	sim_macros.h	

Table 2.2-313: tracks_compute_friction_factor Information.

2.2.6.2.3.2 tracks_compute_slipping_state

This routine calculates the slipping state.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
vel_sign	int	Standard
new_sign	int	Standard
vel_diff	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
slip_state	REAL	the slip state
Calls		
Function	**Where Described**	
tracks_compute_friction_forc e	Section 2.2.6.2.3.1	
abs	sim_macros.h	
sign	sim_macros.h	

Table 2.2-314: tracks_compute_slipping_state Information.

2.2.6.2.3.3 tracks_compute_gravity_load

This function is called by the drive sprocket to compute the load force on the sprocket due to gravitational forces. The convention used is that positive load forces oppose the direction of sprocket drive motion (i.e. negative loading helps, and positive loading opposes).

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
pitch_sin	REAL	sim_types.h
pitch_cos	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
force	REAL	friction force
Calls		
Function	**Where Described**	
kinematics_pitch_sin	Section 2.5.8.5.1	
sqrt	sim_macros.h	
square	sim_macros.h	

Table 2.2-315: tracks_compute_gravity_load Information.

2.2.6.2.3.4 tracks_compute_drag_load

This function is called by the drive sprocket to compute the load force on the sprocket due to drag forces. Since drag forces are always in opposition to the sprocket, this function will always return a positive value.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
drag_force	REAL	sim_types.h
buildup_speed	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
drag_force	REAL	The drag force on the sprocket
Errors		
Error Name	**Reason for Error**	
Track Error	Unknown soil type	
Calls		
Function	**Where Described**	
abs	sim_macros.h	
square	sim_macros.h	

Table 2.2-316: tracks_compute_drag_load Information.

2.2.6.2.3.5 tracks_repair_thrown_tracks

This routine repairs a thrown track.

Calls	
Function	**Where Described**
bigwheel_repair_tracks	Section 2.5.10.10.3

Table 2.2-317: tracks_repair_thrown_tracks Information.

2.2.6.2.3.6 tracks_throw_left_track

This routine throws the left track.

Calls	
Function	**Where Described**
bigwheel_left_track_broken	Section 2.5.10.10.1

Table 2.2-318: tracks_throw_left_track Information.

2.2.6.2.3.7 tracks_throw_right_track

This routine throws the right track.

Calls	
Function	**Where Described**
bigwheel_right_track_broken	Section 2.5.10.10.2

Table 2.2-319: tracks_throw_right_track Information.

2.2.6.2.3.8 tracks_compute_weight

This routine returns the weight of the tank.

Return Values		
Return Value	**Type**	**Meaning**
TANK_WEIGHT	REAL	The weight of the tank

Table 2.2-320: tracks_compute_weight Information.

2.2.6.2.3.9 tracks_compute_real_velocity

This routine returns the tank's velocity.

Return Values		
Return Value	**Type**	**Meaning**
vehicle_actual_velocity	REAL	The velocity of the tank

Table 2.2-321: tracks_compute_real_velocity Information.

2.2.6.2.3.10 tracks_compute_velocity

This routine returns the mean velocity of the tracks.

Return Values		
Return Value	**Type**	**Meaning**
mean_track_velocity	REAL	The mean velocity of the tracks

Table 2.2-322: tracks_compute_velocity Information.

2.2.6.2.3.11 odometer_simul

This routine updates the mileage count for the odometer each tick, informing the stochastic failures module of the mileage.

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

Calls	
Function	**Where Described**
abs	sim_macros.h
controls_odometer_pulse	Section 2.2.2

Table 2.2-323: odometer_simul Information.

2.2.6.2.3.12 tracks_set_initial_distance_km

This routine sets the initial distance by setting the odometer reading to *distance*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
distance	REAL	sim_types.h

Table 2.2-324: tracks_set_initial_distance Information.

2.2.6.2.3.13 vehicle_get_elapsed_km

This routine returns the elapsed distance in kilometers.

Return Values		
Return Value	**Type**	**Meaning**
odometer_reading	REAL	elapsed distance

Table 2.2-325: vehicle_get_elapsed_km Information.

2.2.6.2.3.14 tracks_send_velocities

This routine computes the velocities and sets the odometer accordingly.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
left	REAL	sim_types.h
right	REAL	sim_types.h
torque	REAL	sim_types.h
Calls		
Function	**Where Described**	
odometer_simul	Section 2.2.6.2.3.11	

Table 2.2-326: tracks_send_velocities Information.

2.2.6.2.3.15 tracks_stop_drivetrain

This routime stops the tank by setting the velocities to zero.

2.2.6.2.3.16 tracks_init

This routine initializes the tracks.

Calls	
Function	**Where Described**
bigwheel_veh_init	Section 2.5.10
suspension_params	Section 2.5.6.5.1
fail_init_failure	Section 2.5.4.11.2

Table 2.2-326: tracks_init Information.

2.2.6.2.3.17 tracks_compute_vehicle_force

Parameters		
Parameter	**Type**	**Where Typedef Declared**
force	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
force	REAL	the force exerted by friction
Calls		
Function	**Where Described**	
tracks_compute_friction_force	Section 2.2.6.2.3.1	

Table 2.2-327: tracks_compute_vehicle_force Information.

2.2.6.2.3.18 compute_actual_vehicle_motion

This routine calculates the motion of the tank based on the velocities, load, and friction force.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
left_mps	REAL	sim_types.h
right_mps	REAL	sim_types.h

Calls	
Function	**Where Described**
tracks_stop_drivetrain	Section 2.2.6.2.3.15
tracks_compute_gravity_load	Section 2.2.6.2.3.3
tracs_compute_slipping_state	Section 2.2.6.2.3.2
sound_make_const_sound	Section 2.1.3.1.2
sign	sim_macros.h

Table 2.2-328: compute_actual_vehicle_motion Information.

2.2.6.2.3.19 tell_kinematics

This routine updates the position of the vehicle in Kinematics.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
inc	REAL	sim_types.h
angle	REAL	sim_types.h

Calls	
Function	**Where Described**
collision_rear_collision	Section 2.5.10.4.3
collision_left_collision	Section 2.5.10.4.1
collision_right_collision	Section 2.5.10.4.2
tracks_stop_drivetrain	Section 2.2.6.2.3.15
kinematics_move_vehicle	Section 2.5.8.7.1
kinematics_turn_vehicle	Section 2.5.8.11.1
check_turning_sounds	Section 2.2.6.2.3.26

Table 2.2-329: tell_kinematics Information.

2.2.6.2.3.20 get_current_soil_type

This routine gets the soil type of the current position.

Calls	
Function	**Where Described**
terrain_get_terrain_type	Section 2.5.11.3.1

Table 2.2-330: get_current_soil_type Information.

2.2.6.2.3.21 check_for_thrown_track

This routine checks for thrown tracks.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
cant_sin	REAL	sim_types.h

Calls	
Function	**Where Described**
kinematics_cant_sin	Section 2.5.8.5.3
fail_break_system	Section 2.5.4.8.1
abs	sim_macros.h

Table 2.2-331: check_for_thrown_track Information.

2.2.6.2.3.22 send_track_sound

This routine causes the sound of the tracks to be made.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
track_sound	REAL	sim_types.h
track_speed	REAL	sim_types.h

Calls	
Function	**Where Described**
abs	sim_macros.h
min	sim_macros.h
sound_of_tracks	Section 2.1.3.2.9

Table 2.2-332: sound_of_tracks Information.

2.2.6.2.3.23 send_tracks_outputs

This routine sends tracks data output to the appropriate controls and to the sound system.

Calls	
Function	**Where Described**
send_track_sound	Section 2.2.6.2.3.22
meter_speed_set	Section 2.2.2.3.1
abs	sim_macros.h
suspension_acceleration_is	Section 2.5.6.3.1

Table 2.2-333: send_tracks_outputs Information.

2.2.6.2.3.24 get_dust_cloud

This routine returns the appropriate dust cloud index based on soil type and tracks speed.

Internal Variables		
Internal Variable	Type	Where Typedef Declared
speed	REAL	sim_types.h
Return Values		
Return Value	Type	Meaning
vehDustCloudNone	int	no dust cloud
vehDustCloudSmall	int	small dust cloud
vehDustCloudMedium	int	medium dust cloud
vehDustCloudLarge	int	large dust cloud
Calls		
Function	Where Described	
terrain_get_terrain_type	Section 2.5.11.3.1	

Table 2.2-334: get_dust_cloud Information.

2.2.6.2.3.25 tracks_simul

This routine simulates the tracks.

Calls	
Function	Where Described
get_current_soil_type	Section 2.2.6.2.3.20
check_for_thrown_track	Section 2.2.6.2.3.21
compute_actual_vehicle_moti on	Section 2.2.6.2.3.18
tell_kinematics	Section 2.2.6.2.3.19
send_tracks_outputs	Section 2.2.6.2.3.23
network_set_dust_cloud	Section 2.1.1.3.1.12.1
tracks_get_dust_cloud	Section 2.2.6.2.3.24

Table 2.2-335: send_tracks_outputs Information.

2.2.6.2.3.26 tracks_motion_disabled

This routine returns TRUE if the tracks are disabled and returns FALSE otherwise.

Return Values		
Return Value	Type	Meaning
TRUE	BOOLEAN	tracks disabled
FALSE	BOOLEAN	tracks not disabled

Table 2.2-336: tracks_motion_disabled Information.

2.2.6.2.3.27 check_turning_sounds

This routine is called by tracks to see if the sound of the vehicle turning should be started or stopped.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
angle	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Defined**
sign_old_dir	int	Standard
sign_turn_dir	int	Standard
clamped_angle	REAL	sim_types.h
Calls		
Function	**Where Described**	
sound_make_const_sound	Section 2.1.3.1.2	

Table 2.2-337: check_turning_sounds Information.

2.2.6.2.3.28 tracks_return_slip_state

This routine returns the slip state of the vehicle.

Return Values		
Return Value	**Type**	**Meaning**
slip_state	int	the slip state of the vehicle

Table 2.2-338: tracks_return_slip_state Information.

2.2.6.3 Vehicle Subsystems

The CSU's which provide this functionality are as follows:

m1_elecsys.c
m1_hydrsys.c
m1_vision.c

2.2.6.4.1 m1_elecsys.c

(./simnet/release/src/vehicle/m1/src/m1_elecsys.c [m1_elecsys.c])

The model of the M1's electrical system is based the assumption that charge and voltage are linearly related and that the battery charges linearly over time. While many small components (such as indicator lamps) consume electrical charge, only the major charge consumers are modeled. They consist of the starter, laser range finder, auxiliary hydraulic pump, radio, and intercom for the M1.

When the engine is running, it is assumed that all electrical components are obtaining electrical charge for the alternator/generator, which is modeled as an infinite amount of charge. As long as the engine is running and the alternator/ generator has not failed, all requests for electrical charge are granted. The alternator/generator will also charge up a battery which is less than fully charged. However, if the engine is off or the alternator/generator has failed, the electrical components must obtain electrical charge from the battery.

Each of the modeled electrical components requires a specific amount of electrical charge. When the component is used, the electrical system is queried to see if there is enough charge present to accommodate the request. If so, the stored charge is depleted by the amount requested.

Additional assumptions:

If voltage is less than 18 v, the engine will not start.
If voltage is less than 23 v, the low battery charge light turns on.
The battery takes 15 minutes to charge from 23v to 24v.
Each time through the loop, 1/15 of a sec elapses.

The following assumptions are made for calculating the amount by which the battery is discharged by parts which are not always using electricity (starter, laser, pump) at the time of call. It is determined whether or not there is enough charge left in the battery and returns TRUE or FALSE. Then a timer is set up to linearly discharge the battery over the amount of time it would normally take.

Due to use of the starter:

- it takes 1 second to start
- STARTER_DISCHARGE_DELTA amp-hours are used up over the 1 sec
- the battery is discharged linearly
- the starter is allowed to function 40 times (not including recharging) before the abort light comes on.
- STARTER_DISCHARGE_DELTA = 74 amphours/ NUMBER OF TIMES (40)
- calculations: 15 frames/sec 12 1 sec = 15 frames
  1.85 amp-hours / 15 frames = .12333333333 amp-hours/frame

Due to use of the auxiliary pump:
- a 1 hp pump (not an assumption), 1 hp = 746 watts
- power = vi -> current = 746 watts/24 volts
- i = dq/dt
- dt = 1/15 sec/frame * 1/60 min/sec * 1/60 hr/min = 1/54000 hr/frame
- dq = 746/24 * 1/54000 = 373/648000 amp-hr/frame for linear discharge

Due to use of the radio:
- the radio can be used for 20 hours before the engine can no longer start (battery discharged by 75 amp-hr)
- the radio is always on if master power is on
- the radio uses 3.75 amp-hr/hr
- 3.75 amp-hr/hr * 1/60 hr/min * 1/60 min/sec * 1/15 sec/frame
- dq = 3.75/54000 amp-hr/frame, assuming linear discharge

Due to use of the intercom:
- the intercom lasts 15 hours before the engine can no longer start (battery discharged by 75 amp-hrs)
- the intercom is always on if master power is on
- 75 amp-hours/15 hours * 1/3600 hr/sec = 5/3600 = 1/720 amp-hr/sec
- dq = 1/15 sec/fr * 1/720 amp-hr/sec = 1/10800 amp-hr/fr discharged linearly

Due to use of the laser range finder:
- the laser range finder can be fired 100 times before the engine will not start (discharged 75 amphr)
- 75 / 100 = .75 amp - hr used each time the laser range finder is charged up

The following assumptions are used to determine the voltage shown on the voltmeter when the engine is on:
- voltage put out by the alternator is 27.5 to 28.5 volts
- rpms : low idle - 900
  tac idle - 1313
  max idle - 3100
- assume 900 to 3100 corresponds piecewise linearly to 27.5 to 28.5 volts
- When driving, rpm is generally between 900 and 2000 rpms, therefore the range of volts is 27.5 to 28.2 volts
- From 2000 to 3100 rpms, the range of volts will be 28.2 to 28.5 volts

The following assumptions are used to determine the voltage shown on the voltmeter when the engine is on, but the alternator is broken:
- The engine runs off the battery for 5 minutes. Then the battery drops to the WEAKLY_CHARGED state..
- The engine runs for 5 minutes and then stops. (battery discharged by 75 amp-hrs)
- The intercom is always on if master power is on
- 75 amp-hours/(1/12) hours * 1/3600 hr/sec = 1/4 amp-hr/sec
- dq = 1/15 sec/fr * 1/4 amp-hr/sec = 1/60 amp-hr/fr, discharged linearly

The following assumptions are used to determine the rate for charging the battery:
- If battery drops from 24v to 23v, it takes 15 minutes with the engine running to recharge it back to 24 volts. (0.25 hr/volt)
- Assume the recharge curve is linear
- (engine should have low crank speed if battery <= 23, note that no sound models this right now)

- 15 frames/sec * 60 sec/min * 15 min = 13500 frames to recharge 1v
- 1volt/13500 frames = 7.407407e-5 volts/frame
- 15 frames/sec * 60 sec/min * 3 min = 2700 frames
- 1 volt/2700 frames = .000370370370 volts/frame

The following define the amount by which the battery is discharged when called:

STARTER_DISCHARGE_DELTA
LASER_DISCHARGE_DELTA
AUX_PUMP_DISCHARGE_DELTA

The following define the amount by which the battery is discharged each tick under correct conditions:

STARTER_DISCHARGE_RATE
RADIO_DISCHARGE_RATE
INTERCOM_DISCHARGE_RATE
ENGINE_DISCHARGE_RATE
BATTERY_RECHARGE_RATE

The following define the battery recharge rate:

BATTERY_RECHARGE_RATE
STARTER_DURATION

The following define constants used to determine the voltage of the alternator:

LOW_IDLE
TAC_IDLE
MAX_IDLE
MAX_ALTERNATOR_VOLTAGE
MIN_ALTERNATOR_VOLTAGE

The following define states of the battery in units of charge:

FULLY_CHARGED	
MEDIUM_CHARGED	- the battery low light will turn on.
WEAKLY_CHARGED	- laser, starter, and auxiliary pump are dead.
BATTERY_DEAD	- all lamps are off
LOWEST_BATTERY_VOLTAGE	- lowest reading on voltmeter.
MAX_CHARGE	- maximum charge held in battery
MAX_VOLTAGE	- maximum voltage of battery

The following conversion factors are defined:

V_TO_Q	- voltage to charge
Q_TO_V	- charge to voltage

The following constants for rpms to volts conversion are defined:

M1	- slope of first part of line
M2	- slope of second part of line
VERTEX	- point where slopes change

The following BOOLEAN values for state of the battery are defined:

NEW	
LEAKY	- leaky batteries charge up but hold no charge when a load is applied.

The following are declared:

battery_charge	- current charge in battery
battery_voltage	- current battery voltage
starter_timer_id	- holds id returned by timer
starter_timer_status	- indicates if timer is on
electsys_status	- indicates if electrical system is on
power_status	- indicates if master or turret power is on
battery_status	- LEAKY or NEW
alternator_status	- if BROKEN, use battery

electsys_discharge_battery();
electsys_charge_battery();
electsys_rpms_to_volts();
electsys_handle_leaky_battery();

Includes:
"stdio.h"
"sim_dfns.h"
"timers.h"
"timers_dfn.h"
"libfail.h"
"failure.h"
"m1_engine.h"
"m1_cntrl.h"
"m1_meter.h"

Defines:
ELECTSYS_DEBUG

2.2.6.3.1.1 electsys_simul

This is the primary routine in this module. If the engine is running and *alternator_status* is true, then the engine is running off the alternator rather than the battery. When the engine is running with the alternator, the battery is recharged. The battery discharges when the engine is running off the battery and when the engine is off. Assume that if asked for the battery usage, either the engine is on or the master power is on.

Calls	
Function	**Where Described**
engine_running	Section 2.2.6.2.2.8
meter_volt_set	Section 2.2.2
electsys_rpms_to_volts	Section 2.2.6.3.1.6
electsys_charge_battery	Section 2.2.6.3.1.3
electsys_discharge_hull_battery	Section 2.2.6.3.1.7
controls_low_charge_off	Section 2.2.2
engine_shutoff_switch	Section 2.2.6.2.2.20
timers_get_ticking_status	Section 2.6.3.20.1
timers_free_timer	Section 2.6.3.5.1

Table 2.2-339: electsys_simul Information.

2.2.6.3.1.2 electsys_dead

This routine calls a routine which sets the electrical system to the inactive state.

Calls	
Function	**Where Described**
controls_electsys_dead	Section 2.2.2

Table 2.2-340: electsys_dead Information.

2.2.6.3.1.3 electsys_charge_battery

This routine recharges the tank's battery.

2.2.6.3.1.4 electsys_power_request

This routine is called by the controls routines to check that power can be turned on.

Return Values		
Return Value	**Type**	**Meaning**
WORKING	BOOLEAN	can turn power on

Table 2.2-341: electsys_power_request Information.

2.2.6.3.1.5 electsys_power_off

This routine sets the *power_status* to OFF.

2.2.6.3.1.6 electsys_rpms_to_volts

This routine allows the electrical system meter to reflect the change in rpms of the engine. This routine is only called while the engine is on. This is accomplished by the y = mx + b formula where y is volts and x is rpms. The y values range from 27.5 volts to 28.5 volts while the x values range from 900 rpms to 3100 rpms. The volts map to rpms using the following formula:

volts = 1/2200 (rpms - 900) + 27.5

The ranges are as follws:

rpms	volts
< 900	27.5
900 - 2000	27.5 -28.2
2000 - 3100	28.2 - 28.5
>3100	28.5

Internal Variables		
Internal Variable	Type	Where Typedef Declared
rpms	REAL	sim_types.h
Return Values		
Return Value	Type	Meaning
MIN_ALTERNATOR_VOLTAGE	REAL	the minimum alternator voltage
MAX_ALTERNATOR_VOLTAGE	REAL	maximum alternator voltage
M1*(rpms-900)+27.5	REAL	voltage for rpms between 900 and 2000
M2*(rpms-3100)+28.5	REAL	voltage for rpms between 2000 and 3100
Calls		
Function	Where Described	
engine_get_speed	Section 2.2.6.2.2.11	

Table 2.2-342: electsys_rpms_to_volts Information.

2.2.6.3.1.7 electsys_discharge_battery

This routine discharges the battery by *delta* if if the engine is off.

Parameters		
Parameter	Type	Where Typedef Declared
delta	REAL	sim_types.h
Calls		
Function	Where Described	
controls_low_charge_on	sim_macros.h	
fail_break_system	Section 2.5.4.8.1	

Table 2.2-343: electsys_discharge_battery Information.

2.2.6.3.1.8 electsys_engine_start_request

This routine determines if the battery is sufficiently charged for the engine to be started.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	engine can be started
FALSE	BOOLEAN	engine cannot be started
Calls		
Function	**Where Described**	
elecsys_handle_leaky_battery	Section 2.2.6.3.1.16	
timers_free_timer	Section 2.6.3.5.1	
timers_get_timer	Section 2.6.3.6.1	

Table 2.2-344: electsys_engine_start_request Information.

2.2.6.3.1.9 electsys_aux_pump_request

This routine determines if the battery is sufficiently charged for the auxiliary pump to be operated.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	pump can be operated
FALSE	BOOLEAN	pump cannot be operated
Calls		
Function	**Where Described**	
elecsys_handle_leaky_battery	Section 2.2.6.3.1.16	
elecsys_discharge_battery	Section 2.2.6.3.1.7	

Table 2.2-345: electsys_aux_pump_request Information.

2.2.6.3.1.10 electsys_laser_start_request

This routine determines if there is enough battery power to operate the laser range finder.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	laser can be operated
FALSE	BOOLEAN	laser cannot be operated

Calls	
Function	**Where Described**
elecsys_handle_leaky_batter y	Section 2.2.6.3.1.16
engine_running	Section 2.2.6.2.2.8
elecsys_discharge_battery	Section 2.2.6.3.1.7

Table 2.2-346: electsys_laser_start_request Information.

2.2.6.3.1.11 electsys_get_battery_voltage

This routine returns the battery voltage.

Return Values		
Return Value	**Type**	**Meaning**
battery_voltage	REAL	the battery voltage

Table 2.2-347: electsys_get_battery_voltage Information.

2.2.6.3.1.12 electsys_replace_alternator

This routine repairs the alternator by setting *alternator_status* to WORKING.

2.2.6.3.1.13 electsys_alternator_failure

This routine causes the alternator to fail by setting *alternator_status* to BROKEN. When the alternator fails, the engine runs off the battery.

2.2.6.3.1.14 electsys_recharge_battery

This routine recharges the battery. The battery voltage returns to 24.0 volts.

Calls	
Function	**Where Described**
hydraulic_charge_reborn	Section 2.2.6.4.2.20
controls_elecsys_reborn	Section 2.2.2
controls_low_charge_off	Section 2.2.2

Table 2.2-348: electsys_recharge_battery Information.

2.2.6.3.1.15 electsys_replace_battery

This routine replaces a leaky battery. *battery_status* is set to NEW.

Calls	
Function	**Where Described**
electsys_recharge_battery	Section 2.2.6.3.1.14

Table 2.2-349: electsys_replace_battery Information.

2.2.6.3.1.16 electsys_handle_leaky_battery

This routine handles the case of a leaky battery. A load is applied across a leaky battery. Although the battery will charge, the voltage will drop to WEAKLY_CHARGED once the load has been applied. The low battery charge light is turned on.

Calls	
Function	**Where Described**
controls_low_charge_on	Section 2.2.2

Table 2.2-350: electsys_handle_leaky_battery Information.

2.2.6.3.1.17 electsys_battery_failure

This routine causes the battery to become leaky by setting *battery_status* to LEAKY.

2.2.6.3.1.18 electsys_vars_status

This routine prints the values of the following variables. It is used as a debugging tool.

battery_charge
battery_voltage
starter_timer_id
starter_timer_status
electsys_status

2.2.6.3.1.19 electsys_init

This routine initializes the electrical system for operation.

Calls	
Function	**Where Described**
meter_volt_set	Section 2.2.2
fail_init_failure	Section 2.5.4.11.2

Table 2.2-351: electsys_init Information.

2.2.6.4.2 m1_hydrsys.c
(./simnet/release/src/vehicle/m1/src/m1_hydrsys.c [hydraulics_simul])

The routines used to model the M1's hydraulic system are found in one CSU, m1_hydrsys.c. As with the electrical system model, only specified components actually operate using hydraulic pressure in this model. They are the traversing of the turret, elevation of the gun, opening and closing of the ammo door, and setting of the parking brake. For the ammo door and parking brake, a discrete amount of hydraulic pressure is depleted from the hydraulic reservoir each time the component is accessed. For the turret and gun, hydraulic pressure is depleted from the reservoir every frame that the turret is traversing or the gun is elevating or depressing. The exact amount depends how quickly the turret or gun is moving. If the pressure in the reservoir falls below specified levels, the query to the hydraulic system will fail and the component will fail to operate properly.

The model has both a main hydraulic pump and an auxiliary hydraulic pump. When the engine is running, the main pump is operational, and it is more than sufficient to meet all the hydraulic needs of the vehicle. Therefore, the vehicle responds as though it has an infinite supply of hydraulic pressure because the reservoir is being replenished as quickly as it is being depleted. When the engine if off, however, the auxiliary hydraulic pump is used to replenish the reservoir. It does not accomplish this as quickly as the main pump does so hydraulic pressure may run out. If the pressure drops too low, the operator must wait for the pressure to build up again before using any components requiring hydraulic pressure.

The hydraulic system is modelled as a capacitor. Pressure (psi) is modelled as volts, flow (gpms) is modelled as current, hydraulic const (gal-sec/psi-min) is modelled as capacitance (farads). The following information is used to calculate the hydraulic constant :

- whether using the auxiliary pump
- turret traversal impossible when hydraulic pressure < 900 psi.
- with pressure at 1500 psi, turret may be slewed for 6.0 seconds (turned 150 degrees) before pressure becomes too low.
- takes 45 seconds for hydraulic pressure to go from 500 - 1500 psi.
- K * 1000 psi/45 seconds = 5 gal/min; K = .225 gal-sec/psi-min.
- K = 0.00375 gal/psi
- flow from hydraulic reservoir = K * (1500 - 900) psi/6.0 sec.
  flow = 22.5 gal/min when slewing the turret, but will use these values for other operations (value is for full handle displacement).
- 22.5 gal/min * 1/60 min/sec * 1/15 sec/frame = 0.025 gal/frame
- to calculate max reduction of psi/frame of the reservoir, use
  FLOW_OUT_RATE = 0.025 gal/frame.
- flow = K * dp/dt -> dp = flow /K * dt; where dt = 1 frame
- dp = 0.025 gal/frame / 0.00375 gal/psi * 1 frame = 6.66666667 psi
- system is modelled with only one accumulator containing 800 psi
- regulator drops 100 psi, so hydraulic reservoir must hold at least 900 psi for consistent operation.

The following constants associated with the hydraulic model are defined:

HYDR_CONST	in gal/psi
MAX_FLOW_OUT_RATE	in gal/frame
MAX_PSI_FLOW_OUT	in psi/frame
OPERATIONAL_SPEED	in rpms

The range of the main pump is 1550 - 1700 psi

- Pump turns on if hydraulic reservoir drops below 1550 psi
- Pump turns off if hydraulic reservoir exceeds 1700 psi
- For normal hydraulic system operation, flow rate is:27.1 gpm if engine is idle, 47.4 gpm if engine is at operational speeds.
- flow = K dp/dt
- 27.1 gal/min = HYDR_CONST * dp/1 frame
  dp/fr = 27.1 gal/min / 0.00375 gal/psi * min/60 sec * sec/15 frame
  = 8.0296296296296 psi/frame
- if flow = 47.4 gpm, dp/fr -> 14.0444444444444444 psi/frame

The following constants associated with the engine are defined:

ENGINE_IDLE_PSI_IN_RATE	in psi/frame
ENGINE_OPER_PSI_IN_RATE	in psi/fame

The range of the auxiliary pump is 1150 - 1650 psi
- Pump turns on if hydraulic reservoir drops below 1150 psi
- Pump turns off if hydraulic reservoir exceeds 1650 psi
- max flow rate is 5 gpm

The following constant is defined:

ENGINE_OFF_PSI_IN_RATE	in psi/frame

This code models the actual M1 accurately by including an accumulator and a regulator (although they are not necessary for this code). The following constants are associated with the accumulator:

REGULATOR	in psi
ACC_PRESSURE_MAX	in psi
ACC_WORKING_THRESHOLD	in psi
MAIN_PUMP_ON_VAL	in psi
MAIN_PUMP_OFF_VAL	in psi
AUX_PUMP_ON_VAL	in psi
AUX_PUMP_OFF_VAL	in psi
RESERVOIR_MIN	in psi

The following are defined for use in this modeule:

YES	
NO	
RUN_LIMIT	if counter < 7, slew
STOP_LIMIT	if 7 > ctr < 16, stop
HYDRAULIC_VERBOSE	
OPEN	
CLOSED	
MAIN	
AUX	

The following constants are associated with the ammo door:

AMMO_DOOR_OPEN_DELTA	in psi
AMMO_DOOR_DELTA	in psi, while door is open
AMMO_DOOR_CLOSED_DELTA	in psi
AMMO_DOOR_LIMIT	in psi

The following constants are associated with the parking brake:

PARKING_BRAKE_DELTA	in psi
PARKING_BRAKE_LIMIT	in psi

The following variables are declared for use in this module:

main_pump_status	- working or broken
aux_pump_status	- working or broken
acc	- assume using one main accumulator, keeps track of pressure in accumulator
reservoir	- amount of pressure hydraulic reservoir
hydraulic	- tells if a hydraulic pump is gone
ammo_door_status	- tells if door is OPEN or CLOSED
master_power_status	- tells if master power is ON or OFF
enough_charge	- tells if battery had enough charge last time
slew_jerk_ctr	- allows for jerking during turret slew if pressure is too low
elev_jerk_ctr	- allows for jerking during turret elevation if the pressure is too low

The following routines are declared:

hydraulic_main_pump_fill ()
hydraulic_aux_pump_fill ()
hydraulic_delta_pressure_calc ()
hydraulic_check_acc ()
hydraulic_fraction_flow_rate ()
hydraulic_deplete_reservoir ()

The following are included:

"stdio.h"
"sim_dfns.h"
"sim_types.h"
"libsound.h"
"m1_elecsys.h"
"m1_engine.h"
"m1_sound_dfn.h"

2.2.6.4.2.1 hydraulic_simul

This routine simulates the tanks hydraulic system using the model described above.

Calls	
Function	**Where Described**
engine_running	Section 2.2.6.2.2.8
hydraulic_main_pump_fill	Section 2.2.6.4.2.5
hydraulic_aux_pump_fill	Section 2.2.6.4.2.6
sound_make_const_sound	Section 2.1.3.1.2

Table 2.2-352: hydraulic_simul Information.

2.2.6.4.2.2 hydraulic_init

This routine initializes the hydraulic system. It sets the following variables to their initial values:

reservoir	AUX_PUMP_OFF_VAL
acc	ACC_PRESSURE_MAX
ammo_door_status	CLOSED
master_power_status	OFF
hydraulic	OFF
enough_charge	TRUE
main_pump_status	WORKING
aux_pump_status	WORKING

2.2.6.4.2.3 hydraulic_check_acc

This routine regulates the contents of the accumulator.

2.2.6.4.2.4 hydraulic_deplete_reservoir

This routine decrements the hydraulic resevoir by *delta* psi.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
delta	REAL	sim_types.h
Calls		
Function	**Where Described**	
hydraulic_check_acc	Section 2.2.6.4.2.3	

Table 2.2-353: hydraulic_deplete_resevoir Information.

2.2.6.4.2.5 hydraulic_main_pump_fill

This routine is used to refill the hydraulic reservoir using the main hydraulic pump. It increments the reservoir by the appropriate pressure (psi) per frame. The increment depends on whether or not the engine is at operational speed.

Calls	
Function	**Where Described**
hydraulic_check_acc	Section 2.2.6.4.2.3
engine_get_speed	Section 2.2.6.2.2.11

Table 2.2-354: hydraulic_aux_pump_fill Information.

2.2.6.4.2.6 hydraulic_aux_pump_fill

This routine is used to refill the hydraulic reservoir using the auxiliary hydraulic pump. It increments the hydraulic pressure by the appropriate psi/frame. *enough_charge* keeps track of whether or not there was enough battery charge at the last request. If *enough_charge* = 0, there must be a call from the electrical system letting the hydraulic system know the battery has been recharged.

Calls	
Function	**Where Described**
hydraulic_check_acc	Section 2.2.6.4.2.3
electsys_aux_pump_request	Section 2.2.6.3.1.9

Table 2.2-355: hydraulic_aux_pump_fill Information.

2.2.6.4.2.7 hydraulic_ammo_door_open_request

Each time the ammo door is opened, 2.4 psi is decremented from the reservoir, and 0.1 psi is depleted from the reservoir each tick if the door is left open. After the accumulator has dropped below 750 psi, the ammo door will not work.

Return Values		
Return Value	**Type**	**Meaning**
FALSE	BOOLEAN	ammo door can't open
TRUE	BOOLEAN	ammo door can open
Calls		
Function	**Where Described**	
hydraulic_deplete_reservoir	Section 2.2.6.4.2.4	

Table 2.2-356: hydraulic_ammo_door_open_request Information.

2.2.6.4.2.8 hydraulic_ammo_door_closed

The ammo door is closed by setting ammo_door_status to CLOSED. Each time the ammo door is closed, 2.4 psi is depleted from the reservoir.

Calls	
Function	**Where Described**
hydraulic_deplete_reservoir	Section 2.2.6.4.2.4

Table 2.2-357: hydraulic_ammo_door_closed Information.

2.2.6.4.2.9 hydraulic_parking_brake_on_request

Each time the parking brake is depressed, 2.5 psi is removed from the hydraulic reservoir, even if there is not sufficient pressure to stop the tank. If there is less than 500 psi, the hydraulic reservoir fluid will not be available and will not be removed from the reservoir. The 2.5 psi is removed over one frame. This routine returns TRUE if the brake can be used and FALSE otherwise.

Return Values		
Return Value	**Type**	**Meaning**
FALSE	BOOLEAN	parking brake cannot be used
TRUE	BOOLEAN	parking brake can be used
Calls		
Function	**Where Described**	
hydraulic_deplete_reservoir	Section 2.2.6.4.2.4	

Table 2.2-358: hydraulic_ammo_door_on_request Information.

2.2.6.4.2.10 hydraulic_slew_turret_request

This routine processes a turret slew request. *fraction_to_move* is a value between 0 and 1 and is used to determine how much of the maximum elevation rate the gun will move. MAX speed corresponds to 1. This is used to determine how much of the 22.5 gpm maximum flow rate must be used to drain pressure from the hydraulic reservoir. This routine should be called every tick in which the turret wants to move. This routine returns YES if there is sufficient hydraulic pressure for the turret to move and returns NO otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fraction_to_move	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
YES	BOOLEAN	turret can slew
NO	BOOLEAN	turret can not slew
Calls		
Function	**Where Described**	
hydraulic_fraction_flow_rate	Section 2.2.6.4.2.12	
hydraulic_deplete_reservoir	Section 2.2.6.4.2.4	
hydraulic_delta_pressure_calc	Section 2.2.6.4.2.13	

Table 2.2-359: hydraulic_elevate_turret_request Information.

2.2.6.4.2.11 hydraulic_elevate_gun_request

This routine processes a gun elevation request. *fraction_to_move* is a value between 0 and 1 used to determine how much of the maximum elevation rate the gun will move. MAX speed corresponds to 1. This is used to determine how much of the 22.5 gpm maximum flow rate to use to drain pressure from the hydraulic reservoir. This routine should be called every tick in which the gun wants to move. Assume that the gun elevation takes 1/5 less hydraulic pressure when elevating at MAX speed than it takes to slew turret at MAX speed. This routine returns YES if there is sufficient hydraulic pressure for the gun to move and returns NO otherwise.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fraction_to_move	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
YES	BOOLEAN	gun can elevate
NO	BOOLEAN	gun cannot elevate
Calls		
Function	**Where Described**	
hydraulic_fraction_flow_rate	Section 2.2.6.4.2.12	
hydraulic_deplete_reservoir	Section 2.2.6.4.2.4	
hydraulic_delta_pressure_calc	Section 2.2.6.4.2.13	

Table 2.2-360: hydraulic_elevate_gun_request Information.

2.2.6.4.2.12 hydraulic_fraction_flow_rate

This routine calculates the hydraulic flow rate per frame. It returns a percentage of the maximum possible flow rate per frame.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fraction_to_move	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
fraction_to_move * MAX_FLOW_OUT_RATE	REAL	hydraulic flow rate per frame

Table 2.2-361: hydraulic_delta_pressure_calc Information.

2.2.6.4.2.13 hydraulic_delta_pressure_calc

This routine calculates and returns the change in pressure (psi) of the hydraulic reservoir, given the flow rate per frame (*flow_rate*).

K * dp/dt = flow/dt

dp = flow/K.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
flow_rate	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
flow_rate/HYDR_CONST	REAL	the change in pressure of the hydraulic reservoir

Table 2.2-362: hydraulic_delta_pressure_calc Information.

2.2.6.4.2.14 hydraulic_master_power_on

This routine turns the hydraulic master power on by setting *master_power_status* to ON.

2.2.6.4.2.15 hydraulic_master_power_off

This routine turns the hydraulic master power off by setting *master_power_status* to OFF.

2.2.6.4.2.16 hydraulic_repair_main_pump

This routine repairs the main hydraulic pump by setting *main_pump_status* to WORKING.

2.2.6.4.2.17 hydraulic_repair_aux_pump

This routine repairs the auxiliary hydraulic pump by setting *aux_pump_status* to WORKING.

2.2.6.4.2.18 hydraulic_main_pump_failure

This routine sets the status of *main_pump_status* to BROKEN, causing the main hydraulic pump to fail. A message is printed to the standard output indicating that the pump has failed.

2.2.6.4.2.19 hydraulic_aux_pump_failure

This routine sets the status of *aux_pump_status* to BROKEN, causing the auxiliary hydraulic pump to fail. A message is printed to the standard output indicating that the pump has failed.

2.2.6.4.2.20 hydraulic_charge_reborn

This routine is called from the electrical system routines to inform the hydraulic system that the battery has been recharged.

2.2.6.4.2.21 hydrsys_vars_status

This routine prints the value of the following variables:

hydraulic
resevoir
ammo_door_status
slew_jerk_ctr
elev_jerk_ctr

2.2.6.4.3 m1_vision.c
(./simnet/release/src/vehicle/m1/src/m1_vision.c [m1_vision.c])

The file, m1_vision.c contains routines to break the viewports (turn them off) and fix them by toggling bits in libvflags. In addition, m1_vision.c contains routines to switch between thermal and out the window views for CATC training. There routines only affect simulators which have the proper hardware modifications to do thermal switching. The ability of the cupolas to pitch up and down is also controlled in the vision files.

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"libfail.h"
"failure.h"
"mass_stdc.h"
"libvflags.h"
"m1_vision.h"

Defines:

View Flags	Value
DVRS_LT	VIEWP_5
DVRS_CTR	VIEWP_6
DVRS_RT	VIEWP_7
CMDR_LT	VIEWP_1
CMDR_CTR	VIEWP_2
CMDR_RT	VIEWP_3
CMDRS_VIEWP	(CMDR_LT|CMDR_CTR|CMDR_RT)
DVRS_VIEWP	(DVRS_LT|DVRS_CTR|DVRS_RT)
LDR_VIEWP	(VIEWP_4)
GNR_VIEWP	(VIEWP_0)
CMDR_PITCH_BR_VAL	1
LMR_PITCH_BR_VAL	2
CMDR_BIN_BR_VAL	3

Configuration Viewports	Value
GNR	0
CMRL	1
CMRC	2
CMRR	3
LDR	4
DVRL	5
DVRC	6
DVRR	7

Symbol	Value	
VISION_BLOCK_SELF_REPAIR_TIME	10	minutes

int declarations and initialization:
catc_mode = FALSE

WORD declarations and initialization:

vision_state = OTW_DAY	day or night
gunners_state = OTW_DAY	day, night or thermal
drivers_state = OTW_DAY	day, thermal

2.2.6.4.3.1 set_gunners_state

This routine sets the gunner's view state to *state*.

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

Table 2.2-363: set_gunners_state Information.

2.2.6.4.3.2 set_vision_state

This routine sets the vision state to *state*.

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

Table 2.2-364: set_vision_state Information.

2.2.6.4.3.3 set_drivers_state

This routine sets the driver's view state to *state*.

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

Table 2.2-365: set_drivers_state Information.

2.2.6.4.3.4 vision_cmdrs_pitch

This routine sets branch values and sets the commander's pitch state to *pitch_state*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pitch_state	WORD	mass_stdc.h
Calls		
Function	**Where Described**	
set_br_vals	Section 2.1.2.2.4.7.1	

Table 2.2-366: vision_cmdrs_pitch Information.

2.2.6.4.3.5 vision_loaders_pitch

This routine sets branch values and sets the loader's pitch state to *pitch_state*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pitch_state	WORD	mass_stdc.h
Calls		
Function	**Where Described**	
set_br_vals	Section 2.1.2.2.4.7.1	

Table 2.2-367: vision_loaders_pitch Information.

2.2.6.4.3.6 vision_cmdrs_binoculars

This routine sets branch values and sets the commanders binoculars to NO_BINOC.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
bin_state	WORD	mass_stdc.h
Calls		
Function	**Where Described**	
set_br_vals	Section 2.1.2.2.4.7.1	

Table 2.2-368: vision_cmdrs_binoculars Information.

2.2.6.4.3.7 vision_restore_all_blocks

This routine sets view flags and view modes to restore all views.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
Calls		
Function	**Where Described**	
set_view_flags	Section 2.1.2.2.4.8.1	
set_vmodes	Section 2.1.2.2.4.9.1	

Table 2.2-369: vision_restore_all_blocks Information.

2.2.6.4.3.8 vision_break_all_blocks

This routine clears view flag bits, blackening all screens.

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

Calls	
Function	Where Described
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-370: vision_break_all_blocks Information.

2.2.6.4.3.9 vision_break_gps

This routine clears view flag bits for gunner view, blackening the gunner's primary sight screen.

Calls	
Function	Where Described
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-371: vision_break_gps Information.

2.2.6.4.3.10 vision_break_driver_blocks

This routine clears view flag bits for driver and commander views, blackening the driver's screens.

Calls	
Function	Where Described
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-372: vision_break_driver_blocks Information.

2.2.6.4.3.11 vision_break_driver_center_block

This routine clears view flag bits for driver views, blackening the driver's center screen.

Calls	
Function	Where Described
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-373: vision_break_driver_center_block Information.

2.2.6.4.3.12 vision_break_cmdrs_blocks

This routine clears view flag bits for commander views, blackening the commander's screens.

Calls	
Function	**Where Described**
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-374: vision_break_cmdrs_blocks Information.

2.2.6.4.3.13 vision_break_ldrs_pscope

This routine clears view flag bits for loader views, blackening the loader's periscope screen.

Calls	
Function	**Where Described**
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-375: vision_break_ldrs_pscope Information.

2.2.6.4.3.14 vision_restore_gps

This routine sets view flags, restoring the gunner's primary sight screen.

Calls	
Function	**Where Described**
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-376: vision_restore_gps Information.

2.2.6.4.3.15 vision_restore_driver_blocks

This routine sets view flags, restoring the driver's screens.

Calls	
Function	**Where Described**
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-377: vision_restore_driver_blocks Information.

2.2.6.4.3.16 vision_restore_cmdrs_blocks

This routine sets view flag bits, restoring the commander's screens.

Calls	
Function	**Where Described**
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-378: vision_restore_cmdrs_blocks Information.

2.2.6.4.3.17 vision_restore_ldrs_pscope

This routine sets view flags for the loader's periscope, restoring the screen.

Calls	
Function	**Where Described**
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-379: vision_restore_ldrs_pscope Information.

2.2.6.4.3.18 vision_get_dvr_rt_vp

This routine returns the driver's right viewport state.

Return Values		
Return Value	**Type**	**Meaning**
DVRR	int	deiver's right viewport state.

Table 2.2-380: vision_get_dvr_rt_vp Information.

2.2.6.4.3.19 vision_get_dvr_ctr_vp

This routine returns the driver's center viewport state.

Return Values		
Return Value	**Type**	**Meaning**
DVRC	int	driver's center viewport state

Table 2.2-381: vision_get_dvr_ctr_vp Information.

2.2.6.4.3.20 vision_get_dvr_lt_vp

This routine returns the driver's left viewport state.

Return Values		
Return Value	**Type**	**Meaning**
DVRL	int	driver's left viewport state

Table 2.2-382: vision_get_dvr_lt_vp Information.

2.2.6.4.3.21 vision_get_gnr_vp

This routine returns the gunner's viewport state.

Return Values		
Return Value	**Type**	**Meaning**
GNR	int	gunner's viewport state

Table 2.2-383: vision_get_gnr_vp Information.

2.2.6.4.3.22 vision_set_otw_night_vision

This routine sets the OTW night vision.

2.2.6.4.3.23 vision_set_gunner_white_hot_thermal

This routine sets view mode for gunner to white hot thermal.

Calls	
Function	**Where Described**
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-384: vision_set_gunner_white_hot_thermal Information.

2.2.6.4.3.24 vision_set_driver_white_hot_thermal

This routine sets view mode for driver to white hot thermal.

Calls	
Function	**Where Described**
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-385: vision_set_driver_white_hot_thermal Information.

2.2.6.4.3.25 vision_set_gunner_black_hot_thermal

This routine sets view mode for gunner to black hot thermal.

Calls	
Function	**Where Described**
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-386: vision_set_gunner_black_hot_thermal Information.

2.2.6.4.3.26 vision_set_driver_black_hot_thermal

This routine sets view mode for driver to black hot thermal.

Calls	
Function	**Where Described**
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-387: vision_set_driver_black_hot_thermal Information.

2.2.6.4.3.27 get_catc_mode

This routine returns the status of CATC mode.

Return Values		
Return Value	**Type**	**Meaning**
catc_mode	int	The CATC mode status

Table 2.2-388: get_catc_mode Information.

2.2.6.4.3.28 set_catc_mode

This routine sets CATC mode to TRUE.

2.2.6.4.3.29 get_vision_state

This routine returns the vision state.

Return Values		
Return Value	**Type**	**Meaning**
vision_state	int	The vision state

Table 2.2-389: get_vision_state Information.

2.2.6.4.3.30 vision_set_gunner_no_thermal

This routine sets gunner view flags and view mode for no thermal.

Calls	
Function	**Where Described**
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-390: vision_set_gunner_no_thermal Information.

2.2.6.4.3.31 vision_set_driver_no_thermal

This routine sets driver view flags and view mode for no thermal.

Calls	
Function	**Where Described**
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.2-391: vision_set_driver_no_thermal Information.

2.2.6.4.3.32 toggle_driver_vision_state

This routine toggles driver vision states between OTW day and night or between black and white hot thermal.

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

Calls	
Function	**Where Described**
vision_restore_driver_blocks	Section 2.2.6.4.3.15
vision_set_driver_black_hot_thermal	Section 2.2.6.4.3.26
vision_set_driver_white_hot_thermal	Section 2.2.6.4.3.24

Table 2.2-392: toggle_driver_vision_state Information.

2.2.6.4.3.33 toggle_gunner_vision_state

This routine toggles gunner vision states between OTW day and night or between black and white hot thermal.

Internal Variables		
Internal Variable	Type	Where Typedef Declared
toggle_state	int	Standard
Calls		
Function	Where Described	
vision_set_gunner_no_thermal	Section 2.2.6.4.3.30	
vision_set_gunner_black_hot_thermal	Section 2.2.6.4.3.25	
vision_set_gunner_white_hot_thermal	Section 2.2.6.4.3.23	

Table 2.2-393: toggle_gunner_vision_state Information.

2.2.6.4.3.34 print_view_modes

This routine displays the vision state, gunner's state, and driver's state.

Internal Variables		
Internal Variable	Type	Where Typedef Declared
tmp	pointer to WORD	mass_stdc.h
i	int	Standard
Calls		
Function	Where Described	
get_vmodes	Section 2.1.2.2.4.5.1	

Table 2.2-394: print_view_modes Information.

2.2.6.4.3.35 vision_init

This routine initializes failures in the vision system.

Calls	
Function	Where Described
fail_init_failure	Section 2.5.4.11.2

Table 2.2-395: vision_init Information.

2.2.6.4.4 m1_thermal.c

Includes:
- "stdio.h"
- "sim_dfns.h"
- "sim_types.h"
- "sim_macros.h"
- "mass_stdc.h"
- "dgi_stdg.h"
- "sim_cig_if.h"
- "libmsg.h"
- "pro_sim.h"
- "timers_dfn.h"
- "timers.h"
- "m1_vision.h"
- "m1_comm_mx.h"
- "m1_gunn_mx.h"
- "m1_thermal.h"

Defines:

Thermal Flags	Value
THERMAL_STATE_OFF	0
THERMAL_STATE_COOLING	1
THERMAL_STATE_READY	2
THERMAL_STATE_ON	3
THERMAL_STATE_WARMING	4
THERMAL_TIMER_DONE	1
THERMAL_TIMER_NOT_DONE	0
THERMAL_3X	0
THERMAL_10X	1
THERMAL_WHITE_HOT	0
THERMAL_BLACK_HOT	1
THERMAL_COOLDOWN_DELAY	200
THERMAL_WARMUP_DELAY	20

int declarations:
- therm_state
- therm_mode_switch
- therm_cool_down_state
- therm_warm_up_state
- therm_shutter_closed
- therm_polarity
- therm_mag
- thermal_cool_down_timer
- thermal_warm_up_timer

Procedure declarations:
- start_timing_cooldown_delay()
- start_timing_warmup_delay()
- turn_on_gunners_thermal_view()
- turn_off_gunners_thermal_view()
- stop_cooldown_timer()

stop_warmup_timer()
thermal_cooldown_timeout_check()
thermal_warmup_timeout_check()

2.2.6.4.4.1 thermal_init

This routine initializes the thermal and thermal cooldown states.

2.2.6.4.4.2 thermal_simul

This routine simulates the thermal vision system.

Calls	
Function	**Where Described**
thermal_cooldown_timeout_check	Section 2.2.6.4.4.19
thermal_warmup_timeout_check	Section 2.2.6.4.4.20

Table 2.2-396: thermal_simul Information.

2.2.6.4.4.3 thermal_mode_on

This routine turns on the thermal mode.

Errors	
Error Name	**Reason for Error**
ILLEGAL THERMAL STATE	The variable therm_state has an unexpected value.
Calls	
Function	**Where Described**
start_timing_cooldown_delay	Section 2.2.6.4.4.13
turn_on_gunners_thermal_view	Section 2.2.6.4.4.15

Table 2.2-398: thermal_mode_on Information.

2.2.6.4.4.4 thermal_mode_standby

This routine sets the thermal mode to standby.

Errors	
Error Name	**Reason for Error**
ILLEGAL THERMAL STATE	The variable therm_state has an unexpected value.
Calls	
Function	**Where Described**
start_timing_cooldown_delay	Section 2.2.6.4.4.13
turn_off_gunners_thermal_view	Section 2.2.6.4.4.16

Table 2.2-399: thermal_mode_standby Information.

2.2.6.4.4.5 thermal_mode_off

This routine turns off the thermal mode.

Errors	
Error Name	**Reason for Error**
ILLEGAL THERMAL STATE	The variable therm_state has an unexpected value.
Calls	
Function	**Where Described**
stop_cooldown_timer	Section 2.2.6.4.4.17
start_timing_warmup_delay	Section 2.2.6.4.4.14
turn_off_gunners_thermal_view	Section 2.2.6.4.4.16

Table 2.2-400: thermal_mode_off Information.

2.2.6.4.4.6 thermal_white_hot

This routine sets the gunner's vision to hot thermal objects appearing white.

Calls	
Function	**Where Described**
vision_set_gunner_white_hot_thermal	Section 2.2.6.4.3.23

Table 2.2-401: thermal_white_hot Information.

2.2.6.4.4.7 thermal_black_hot

This routine sets the gunner's vision to hot thermal objects appearing black.

Calls	
Function	**Where Described**
vision_set_gunner_black_hot thermal	Section 2.2.6.4.3.25

Table 2.2-402: thermal_black_hot Information.

2.2.6.4.4.8 thermal_3x

This routine sets the gunner's primary sight to 3X magnification.

Calls	
Function	**Where Described**
cig_gps_mag_3x	Section 2.1.2.2.6.4

Table 2.2-403: thermal_3x Information.

2.2.6.4.4.9 thermal_10x

This routine sets the gunner's primary sight ot 10X magnification.

Calls	
Function	**Where Described**
cig_gps_mag_10x	Section 2.1.2.2.6.4

Table 2.2-404: thermal_10x Information.

2.2.6.4.4.10 thermal_view_on

This routine returns TRUE if the thermal state is on, FALSE otherwise.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	Thermal state is on.
FALSE	int	Thermal state is off.

Table 2.2-405: thermal_view_on Information.

2.2.6.4.4.11 thermal_shutter

This routine switches to the thermal vision system.

Errors	
Error Name	**Reason for Error**
ILLEGAL THERMAL STATE	The variable therm_state has an unexpected value.
Calls	
Function	**Where Described**
vision_break_gps	Section 2.2.6.4.3.9
turn_on_gunners_thermal_view	Section 2.2.6.4.4.15

Table 2.2-406: thermal_shutter Information.

2.2.6.4.4.12 thermal_clear

This routine clears the thermal view.

Errors	
Error Name	**Reason for Error**
ILLEGAL THERMAL STATE	The variable therm_state has an unexpected value.
Calls	
Function	**Where Described**
turn_off_gunners_thermal_view	Section 2.2.6.4.4.16
vision_restore_gps	Section 2.2.6.4.3.14

Table 2.2-407: thermal_clear Information.

2.2.6.4.4.13 start_timing_cooldown_delay

This routine starts the cooldown timer.

Calls	
Function	**Where Described**
timers_get_timer	Section 2.6.3.6.1

Table 2.2-408: start_timing_cooldown_delay Information.

2.2.6.4.4.14 start_timing_warmup_delay

This routine starts the warmup timer.

2.2.6.4.4.15 turn_on_gunners_thermal_view

This routine turns on the gunner's thermal view.

Errors	
Error Name	**Reason for Error**
ILLEGAL THERMAL POLARITY	The variable therm_polarity has an unexpected value.
ILLEGAL THERMAL MAGNITUDE	The variable therm_mag has an unexpected value.
Calls	
Function	**Where Described**
vision_set_gunner_white_hot thermal	Section 2.2.6.4.3.23
vision_set_gunner_black_hot thermal	Section 2.2.6.4.3.25
cig_gps_mag_3x	Section 2.1.2.2.6.4
cig_gps_mag_10x	Section 2.1.2.2.6.4

Table 2.2-409: turn_on_gunners_thermal_view Information.

2.2.6.4.4.16 turn_off_gunners_thermal_view

This routine turns off the gunner's thermal view.

Calls	
Function	**Where Described**
vision_break_gps	Section 2.2.6.4.3.9
get_non_thermal_mag	Section
cig_gps_mag_3x	Section 2.1.2.2.6.4
cig_gps_mag_10x	Section 2.1.2.2.6.4
vision_set_gunner_no_therm al	Section 2.2.6.4.3.30

Table 2.2-410: turn_off_gunners_thermal_view Information.

2.2.6.4.4.17 stop_cooldown_timer

This routine stops the cooldown timer.

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

Table 2.2-411: stop_cooldown_timer Information.

2.2.6.4.4.18 stop_heatup_timer

This routine is a null stub.

2.2.6.4.4.19 thermal_cooldown_timeout_check

This routine checks for the thermal cooldown timer timing out.

Calls	
Function	**Where Described**
timers_get_timeout_edge	Section 2.6.3.22.1
controls_thermal_ready_light_on	Section 2.2.2
turn_on_gunners_thermal_view	Section 2.2.6.4.4.15

Table 2.2-412: thermal_cooldown_timeout_check Information.

2.2.6.4.4.20 thermal_warmup_timeout_check

This routine updates the thermal warmup timer and, if the warmup delay has elapsed, sets the appropriate thermal variables.

Calls	
Function	**Where Described**
controls_thermal_ready_light_off	Section 2.2.2

Table 2.2-413: thermal_warmup_timeout_check Information.

2.2.7 Network Interactions

The structure of the Network Interactions CSC is shown below.

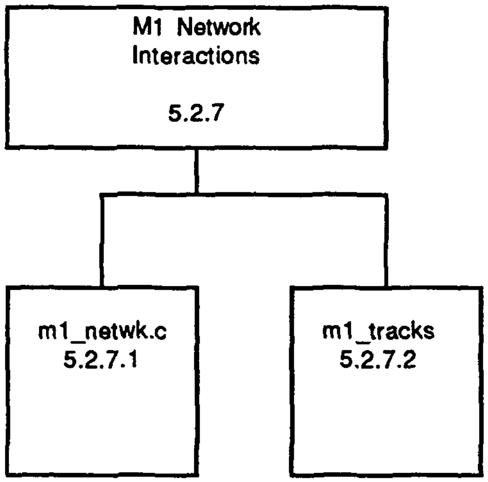

Figure 2.2-8: Structure of the M1 Network Interactions CSC.

Once each frame, the simulation must send an update packet on its own status if that has changed significantly in the current frame. 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 to fill in vehicle-specific fields. Routines are called in m1_tracks.c to determine what size dust cloud, if any, should be reported.

The status of internal subsystems are reported every 30 seconds (for reconstitution and data collection purposes). Routines for sending this status are found in m1_network.c.

The simulation host also sends equipment status packets every 30 seconds, reporting ambient temperature, power supply voltages, and other hardware specific information. The routines for reporting on hardware status are found in the vehicle specific files m1_network.c.

The following CSU provides this functionality:

m1_network.c

2.2.7.1 m1_network.c
(/simnet/release/src/vehicle/m1/m1_network.c [m1_network.c])

Includes:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"pro_data.h"
"pro_sim.h"
"pro_mgmt.h"
"pro_size.h"
"status_m1.h"
"net/network.h"
"libnetwork.h"
"libkin.h"
"libhull.h"
"libfail.h"
"libturret.h"
"libapp.h"
"librva.h"
"m1_status.h"
"m1_engine.h"
"m1_ammo.h"
"m1_elecsys.h"
"m1_fuelsys.h"
"m1_tracks.h"

Defines:

Symbols	Value	
THRESHOLD_FILE	"/simnet/vehicle/m1/data/m1thresh.d"	if _GT_ defined
THRESHOLD_FILE	"/simnet/vehicle/m1/data/m1_thresh.d"	if _GT_ not defined

2.2.7.1.1 send_equipment_status

This routine sends an equipment status PDU over the network.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
pdu	ManagementPDU	p_data.h
pkt	pointer to register EquipStatusVariant	p_mgmt.h

Calls	
Function	**Where Described**
network_get_simulator_type	Section 2.1.1.3.1.19.1
what_is_voltage12P	Section 2.1.5.2.1
what_is_voltage12N	Section 2.1.5.2.2
what_is_voltage5	Section 2.1.5.2.3
what_is_temperature	Section 2.1.5.2.4
is_host_healthy	Section
is_cig_healthy	Section
is_sound_healthy	Section
is_driver_healthy	Section
is_turret_healthy	Section
is_ammo_healthy	Section
network_fill_hdr_send_mgmt_pkt	Section 2.1.1.3.1.42.7
PRO_MGMT_EQUIP_STATUS_SIZE	p_size.h

Table 2.2-414: send_equipment_status Information.

2.2.7.1.2 fill_vehicle_spec_status

This routine fills an M1 vehicle specific status packet.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to register VehicleStatusVariant	p_data.h

Calls	
Function	**Where Described**
engine_get_max_power	Section 2.2.6.2.2.14
electsys_get_battery_voltage	Section 2.2.6.3.1.11
fuel_level_left	Section 2.2.5.2.16
fuel_level_right	Section 2.2.5.2.17
fuel_level_rear	Section 2.2.5.2.15
ammo_get_ready_apds_quantity	Section 2.2.5.1.38
ammo_get_hull_apds_quantity	Section 2.2.5.1.36
ammo_get_ready_heat_quantity	Section 2.2.5.1.37
ammo_get_semi_heat_quantity	Section 2.2.5.1.33
ammo_get_hull_heat_quantity	Section 2.2.5.1.35

Table 2.2-415: fill_vehicle_spec_status Information.

2.2.7.1.3 fill_vehicle_spec_appearance

This routine fills a vehicle specific appearance packet.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to register VehicleAppearanceVariant	p_sim.h

Calls	
Function	**Where Described**
turret_get_network_azimuth	Section 2.5.5.2.15
turret_get_network_elevation	Section 2.5.5.2.14

Table 2.2-416: fill_vehicle_spec_appearance Information.

2.2.7.1.4 network_process_activation_parameters

This routine processes activation parameters: sets failures, sets up tracks, sets vehicle status, sets ammo status, and sets fuel system.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
p	pointer to VehicleStatus	status.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
status	pointer to SIMNET_M1_Status	stat_m1.h
Calls		
Function	**Where Described**	
fail_set_subsys	Section 2.5.4.14.2	
sfail_maintenance_condition	Section 2.5.4.26.2	
tracks_set_initial_distance_km	Section	
ammo_init_ammo_racks	Section 2.2.5.1.5	
fuel_init_tanks	Section 2.2.5.2.1	

Table 2.2-417: network_process_activation_parameters Information.

2.2.7.1.5 app_init

This routine initializes thresholds.

Errors	
Error Name	**Reason for Error**
Network: couldn't init thresholds	A call to network_init_thresholds returned a zero.
Calls	
Function	**Where Described**
network_init_thresholds	Section 2.1.1.3.1.66.5

Table 2.2-418: app_init Information.

2.2.7.1.6 veh_spec_activate_time

This routine returns 60.

Return Values		
Return Value	**Type**	**Meaning**
60	int	N/A

Table 2.2-419: veh_spec_activate_time Information.

2.3 M2 Vehicle Simulation Functions

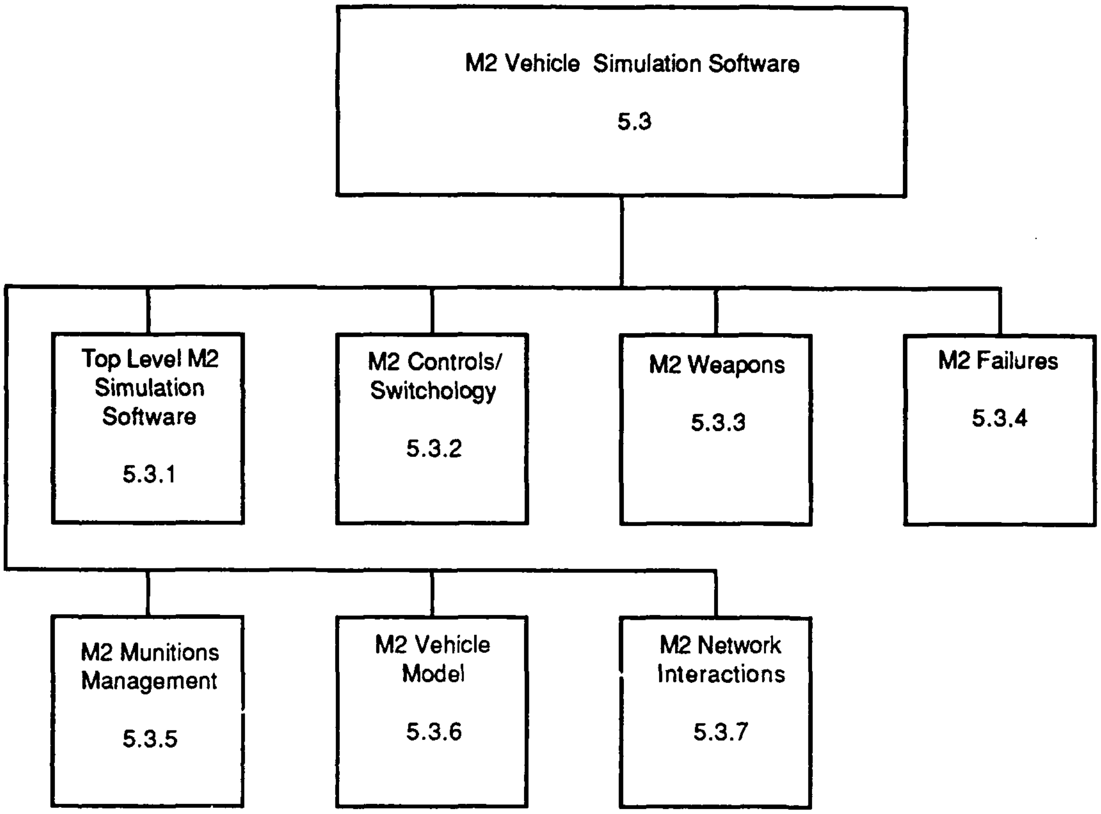

Figure 2.3-1: Structure of the M2 Vehicle Simulation Software CSC.

A distributed vehicle simulation performs a set of tasks each frame. It checks the state of its controls, updates lights and meters, checks its failure/repair model, manages munitions, models its hull (and turret) subsystems, does kinematics/dynamics, simulates weapons, and communicates with the outside world. This cycle of tasks is referred to below as the main simulation loop. The second level CSC's that make up this CSC are as follows:

Top Level M2 Simulation Software
M2 Controls / Switchology
M2 Weapons
M2 Failures
M2 Munitions Management
M2 Vehicle Model
M2 Network Interactions

2.3.1 M2 Top Level Software

In the SIMNET vehicle simulations, the main simulation loop is found in libmain. This loop, executed one each frame, invokes generic functions to perform tasks common to the M1 and M2 simulations. It also invokes a vehicle-specific routine which is defined by the individual simulations in m2_main.c. The one CSU associated with this CSC is therefore m2_main.c.

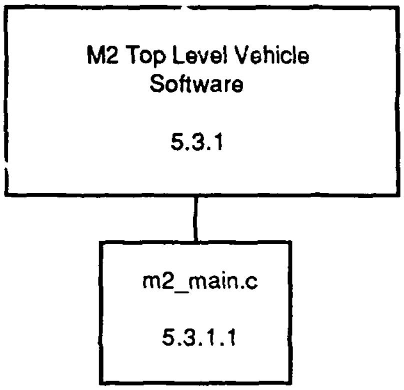

Figure 2.3-2: Structure of the M2 Top Level Vehicle Software CSC.

2.3.1.2 m2_main.c

(/simnet/release/src/vehicle/m1/src/m2_main.c [m2_main.c])

This file contains routines used in the SIMNET simulation of the M2 Bradley Fighting Vehicle.

Includes:

"stdio.h"	"ctype.h"
"signal.h"	"sim_dfns.h"
"sim_macros.h"	"sim_types.h"
"simstdio.h"	"mass_stdc.h"
"dgi_stdg.h"	"sim_cig_if.h"
"pro_assoc.h"	"pro_sim.h"
"status.h"	"status_m2.h"
"veh_type.h"	"mun_type.h"
"failure.h"	"fifo_dfn.h"
"fifo.h"	"bigwheel.h"
"libterrain.h"	"libkin.h"
"libfail.h"	"libcig.h"
"bbd.h"	"libhull.h"
"libidc.h"	"libmain.h"
"libmem.h"	"libmsg.h"
"libnetwork.h"	"librepair.h"
"librva.h"	"libsusp.h"
"libturret.h"	"libsound.h"
"libmap.h"	"m2_alpha.h"
"m2_ammo.h"	"m2_bcs.h"
"m2_cig.h"	"m2_cntrl.h"
"m2_cntrllr.h"	"m2_cons.h"
"m2_cupola.h"	"m2_dtrain.h"

"m2_engine.h" "m2_firectl.h"
"m2_fuelsys.h" "m2_handles.h"
"m2_isu.h" "m2_keybrd.h"
"m2_launcher.h" "m2_meter.h"
"m2_pots.h" "m2_ptrain.h"
"m2_ramp.h" "m2_repair.h"
"m2_ resupp.h" "m2_slope.h"
"m2_sound.h" "m2_status.h"
"m2_weapons.h" "m2_turret.h"
"m2_vision.h" "timers.h"
"dtad.h" "status.h"
"ser_status.h"

The following are declared:
debug
print_overruns
butterfly_silent_mode
reboot_on_shutdown
initial_bbd[]
exit() - if SIMBFLY is not defined

The following are declared for the '-p' switch:
init_activ
initial_activation

The following is defined:
PARS_FILE

2.3.1.1.1 silent_mode_on

This routine sets *butterfly_silent_mode* to TRUE.

2.3.1.1.2 silent_mode_off

This routine sets *butterfly_silent_mode* to FALSE.

2.3.1.1.3 print_help

This routine prints out data for the M2 simulation.

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

Table 2.3-1: print_help Information.

2.3.1.1.4 print_veh_logo

This routine prints a logo for the M2 vehicle.

2.3.1.1.5 veh_spec_startup

This routine sets up the network and CIG interfaces at startup.

Calls	
Function	**Where Described**
network_set_simulator_type	Section 2.1.1.3.1.53.1
use_cig_reconfig_startup	Section
cig_set_view_config_file	Section 2.1.2.2.2.23.1
get_vconfig_file1	Section 2.5.1.2.2
map_vehicle_file_read	Section 2.6.11.5.1
get_veh_map_file	Section 2.5.1.2.5
map_read_asid_file	Section 2.6.11.4.1
get_asid_map_file	Section 2.5.1.2.4
map_file_read	Section 2.6.11.3.1
get_ammo_map_file	Section 2.5.1.2.6
keybrd_init	Section 2.1.6.3
failure_init	Section 2.3.4.1.1
map_get_damage_files	Section 2.6.11.1.1

Table 2.3-2: veh_spec_startup Information.

2.3.1.1.6 veh_spec_idle

This routine is called while the simulator is in the IDLE state.

Calls	
Function	**Where Described**
status_simul	Section 2.1.5.3
keyboard_simul	Section 2.1.6.3
io_simul_idle	Section 2.1.2.2.5.1.2
process_activate_request	Section 2.1.1.3.2.1.1
network_get_exercise_id	Section 2.1.1.3.1.16.1

Table 2.3-3: veh_spec_idle Information.

2.3.1.1.7 veh_spec_init

Order dependent initializations are performed for all of the M2's subsystems while the simulator is in the SIMINIT state.

Calls	
Function	**Where Described**
vision_init	Section 2.3.6.3.2
cupola_init	Section 2.3.6.1.2
sound_init	Section 2.1.3.3.4
alpha_init	Section 2.3.2.3.1
status_preset	Section 2.1.5.3
ammo_init	Section 2.3.5.1.1
controls_fsm_init	Section 2.3.2.2.1
controls_hnp_init	Section 2.3.2
controls_mpc_init	Section 2.3.2.1.2
controls_tnp_init	Section 2.3.2
controls_tpc_init	Section 2.3.2.1.3
bcs_init	Section 2.3.3.1
resupply_init	Section 2.3.5.3.28
meter_init	Section 2.3.2.3.3
electsys_init	Section 2.3.6.3.1.40
firectl_init	Section 2.3.2.2.3
isu_init	Section 2.3.6.3.4
powertrain_init	Section 2.3.6.2.1.1
handles_init	Section 2.3.2.2.2
weapons_init	Section 2.3.3.2
controls_edge_init	Section 2.3.2
app_init	Section 2.3.7.1.5
config_pos_init2	Section 2.1.2.2.2.24.2
kinematics_get_o_to_h	Section 2.5.8.2.4
kinematics_get_w_to_h	Section 2.5.8.2.1
init_brow_pad_state	Section 2.1.2.2.7.3

Table 2.3-4: veh_spec_init Information.

2.3.1.1.8 veh_spec_simulate

This routine calls the routines which simulate the various functins of the M2's subsystems on a tick by tick basis.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
start	long	Standard
end	long	Standard

Calls	
Function	**Where Described**
status_simul	Section 2.1.5.3
keyboard_simul	Section 2.1.6.3
sound_simul	Section 2.1.3.3.6
controls_simul	Section 2.3.2
handles_simul	Section 2.3.2.2.2
ammo_simul	Section 2.3.5.1.2
resupply_simul	Section 2.3.5.3.29
electsys_simul	Section 2.3.6.3.1.8
firectl_simul	Section 2.3.2.2.3
isu_simul	Section 2.3.6.3.4
fuel_simul	Section 2.3.5.2.3
powertrain_simul	Section 2.3.6.2.1.2
bcs_simul	Section 2.3.3.1
weapons_simul	Section 2.3.3.2
ramp_simul	Section 2.3.6.1.3.2
launcher_simul	Section 2.3.6.1.4.2
slope_simul	Section 2.3.2.3.6
cupola_simul	Section 2.3.6.1.2

Table 2.3-5: veh_spec_simulate Information.

2.3.1.1.9 veh_spec_stop

This routine is called while the simulator is in the SIMSTOP state. The IDC and sound system hardware are reinitialized.

Calls	
Function	**Where Described**
idc_init	Section 2.1.4.1.1.24.1
sound_init	Section 2.1.3.2.4
vision_break_all_blocks	Section 2.3.6.3.2

Table 2.3-6: veh_spec_stop Information.

2.3.1.1.10 veh_spec_exit

This routine is called while the simulator is in the SIMEXIT state. Simulation statistics are printed and the network connection is closed.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
num_ticks	int	Standard
Calls		
Function	**Where Described**	
alpha_reset	Section 2.3.2.3.1	
keyboard_exit_gracefully	Section 2.1.6.3	
timers_get_current_time	Section 2.6.3.2.1	
timers_get_current_tick	Section 2.6.3.1.1	
timers_elapsed_milliseconds	Section 2.6.3.10.1	
network_print_statistics	Section 2.1.1.3.2.16.1	
net_close	Section 2.20.2.3.1 in MCC CSCI SDD	

Table 2.3-7: veh_spec_exit Information.

2.3.1.1.11 main

This routine loops through the M2 simulation once each frame. The generic simulation routines in "libmain" are called by this routine.

The parameters are read and parsed:

case -a	Not used.
case -b	Bumper numbers are used.
case -c	Keyboard use cupola.
case -d	Debugging is turned on.
case -D	Debugging for static vehicles is enabled.
case -e	The ethernet is closed.
case -E	The exercise ID is set.
case -F	cfail debug is on.
case -g	The CIG isn't using graphics.
case -h	Print help.
case -?	Print help.
case -k	The key ard is used.
case -n	Vebose mode is used.
case -o	Overrun printing is enabled.
case -p	The simulator is started in stand alone mode. The simulator acts as if it has received an activation packet from the MCC. This segment of code is similar to that used by the MCC to activate a simulator.
case -P	Proximity list debugging is enabled.
case -r	Initial reticle values are set.
case -s	Sound is not used.
case -t	Use the database override named.
case -T	The DED names are set.
case -v	Terrain verbose mode is enabled.
case -V	The voltmeter is disabled.
case 1	The CIG mask and device are set.

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

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
initial_heading	float	Standard
status	pointer to SIMNET_M2_Status	status.h
gp	pointer to GroundVehicleSubsystems	status.h
unit	pointer to OrganizationalUnit	basic.h
status_bits	unsigned int	Standard

Calls	
Function	**Where Described**
enter_gracefully	Section 2.5.1.1.1
network_set_exercise_id	Section 2.1.3.1.49.1
main_read_pars_file	Section 2.5.1.2.1
use_bumper_numbers	
keyboard_use_cupola	Section 2.1.6.3
use_static_debug	Section
network_dont_really_open_up_ethernet	Section 2.1.1.3.2.14.1
network_set_exercise_id	Section 2.1.1.3.1.49.1
cfail_debug_on	Section 2.5.4.2.1
cig_not_using_graphics	Section 2.1.2.2.1.5.1
print_help	Section 2.3.1.1.3
keyboard_really_use	Section 2.1.6.3
v_pkt_verbose_mode	Section 2.1.1.3.1.16.1
rva_turn_debug_on	Section 2.5.12.2.1
idc_set_reticle_init_val	Section 2.1.4.1.3.5
sound_dont_use	Section 2.1.3.3.5
cig_use_database_0verride_named	Section 2.1.2.2.1.16.1
isalpha	Section 2.3.2.3.1
set_ded_name	Section 2.1.2.2.1.8.2
terrain_verbose_mode_on	Section 2.5.11.9.1
electsys_voltmeter_disabled	Section 2.3.6.3.1.39
set_cig_dev	Section 2.1.2.2.1.26.1
set_cig_mask	Section 2.1.2.2.2.114.1
sim_state_startup	Section 2.5.1.1.5
simulation_state_machine	Section 2.5.1.1.13
get_default_db_name	Section 2.5.1.2.15
get_default_db_version	Section 2.5.1.2.16

Table 2.3-8: main Information.

2.3.1.1.12 reconstitute_vehicle

This routine reconstitues a vehicle by sending an activate request sent to the MCC.

Calls	
Function	**Where Described**
process_activate_request	Section 2.1.1.3.2.1.1
network_get_exercise_id	Section 2.1.1.3.1.16.1

Table 2.3-9: reconstitute_vehicle Information.

2.3.2 M2 Controls/Switchology

A large portion of the code generated for specific vehicles exists in this section. Since every vehicle has its own complement of differing controls and indicators, these files exist in vehicle specific code. This second level CSC is further broken down into the following third level CSC's:

Low Level Control Handling
Finite State Machines
Specialized Output Devices

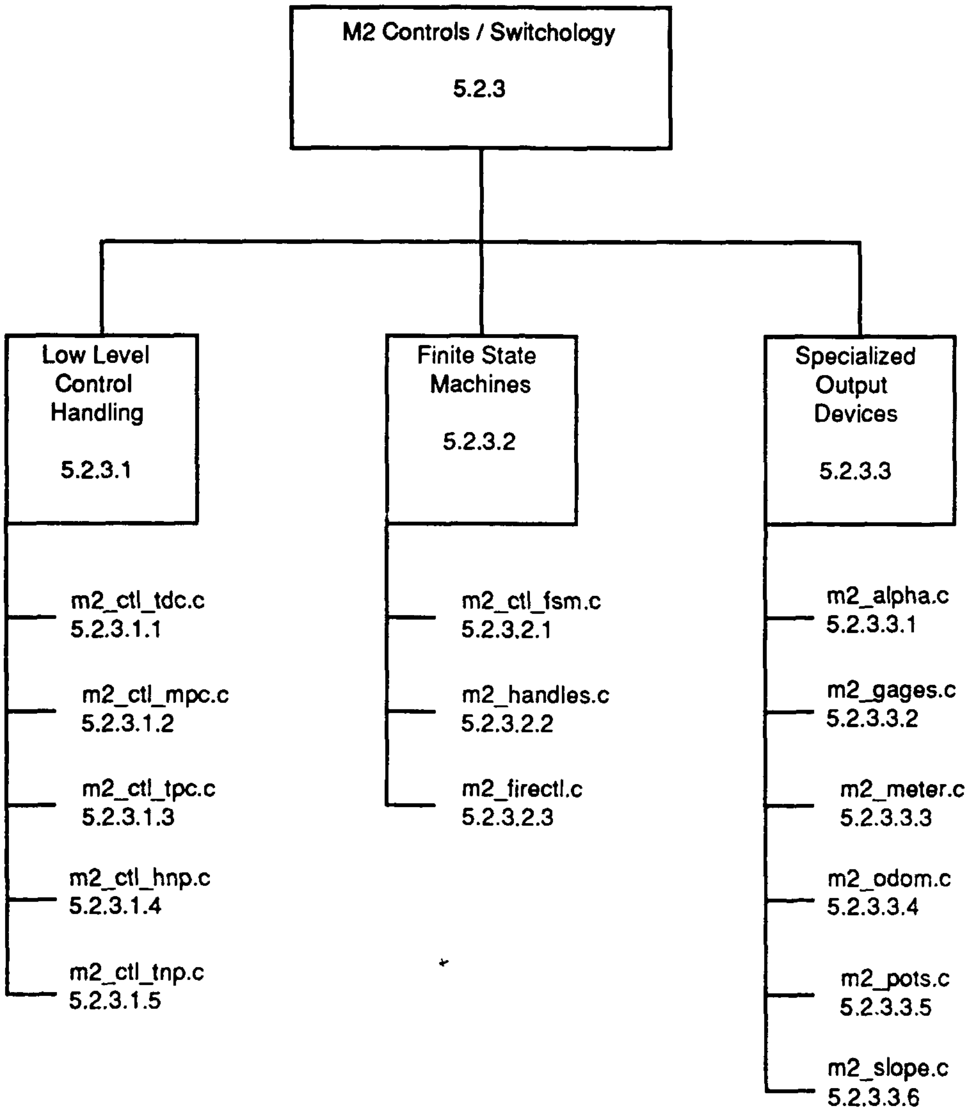

Figure 2.3-3: Structure of the M2 Controls / Switchology CSC.

2.3.2.1 Low Level Control Handling

Since the simulation is interested in transitions of a control, the past state of a control must be maintained and updated each tick of the simulation. Therefore, a state variable is defined for each control in a vehicle simulation. The controls are grouped into separate files based on the "power state". The "power state" of a control is defined as the power requirements to make the control active.

To increase efficiency, the past state of a control is only maintained when a control is active. The last three letters of the files listed above are the initials of the power states:

mpc: master power controls
npc: no power controls
tpc: turret power controls
tnp: turret no power
hnp: hull no power
tdc: turret drive controls

2.3.2.1.1 m2_ctl_tdc.c
(/simnet/release/src/vehicle/m1/src/m2_ctl_npc.c [m2_ctl_tdc.c])

This CSU provides the turret drive controls interface for the M2 simulator. This module includes the following files:

"stdio.h"	"m2_ctl_df.h"
"sim_types.h"	"m2_cntrl.h"
"sim_dfns.h"	"m2_driv_pn.h"
"sim_macros.h"	"m2_turr_pn.h"
"timers_dfn.h"	"m2_driv_mx.h"
"timers.h"	"m2_turr_mx.h"
"libidc.h"	"m2_pots.h"
"libidc_dfn.h"	"m2_handles.h"
"libmem.h"	"m2_weapons.h"
"libmem_dfn.h"	"m2_turreet.h"
"libnetwork.h"	"m2_sound.h"

This file defines the following parameters for use in controlling the turret drive mechanisms:

Variables	Type	Where typedef declared
real_gunn_trav_val	REAL	sim_types.h
real_gunn_elev_val	REAL	sim_types.h
real_comm_trav_val	REAL	sim_types.h
real_comm_elev_val	REAL	sim_types.h
hex_gunn_trav_val	int	Standard
hex_gunn_elev_val	int	Standard
hex_comm_trav_val	int	Standard
hex_comm_elev_val	int	Standard
turret_stab_val	int	Standard
gunner_palm_val	int	Standard
gunner_fast_slew_val	int	Standard
gunner_trigger_val	int	Standard
commander_palm_val	int	Standard
commander_fast_slew_val	int	Standard
commander_trigger_val	int	Standard

Table 2.3-10: m2_ctl_tdc.c variables.

This file defines the routines listed in the table below.

Initialization Routines:

These routines initialize all starting values for the gunner and commander controls. If these controls are on, they are initialized to either off or zero depending on the initial state.

Name	Type of routine
controls_tdc_init ()	void
controls_gunner_traverse_init()	void
controls_gunner_elevation_init ()	void
controls_commander_traverse_init()	void
controls_commander_elevation_init()	void
controls_turret_stab_init()	void
controls_gunner_palm_init()	void
controls_gunner_fast_slew_init()	void
controls_gunner_trigger_init()	void
controls_commander_palm init()	void
controls_commander_fast_slew_init()	void
controls_commander_trigger_init()	void

Table 2.3-11: Initialization Routines.

Turret Drive Routines:

The following routines control the turret drive controls by checking the control state of the turret drive gunner and commander traverse, elevation, stability, slew, trigger values, and palm values. If these values are out of acceptable range as determined by internal routines called by these routines, an error message will appear on the appropriate gunner or commander terminal screen.

Name	Type of routine
controls_gunner_traverse_check()	void
controls_gunner_elevation_check()	void
controls_commander_traverse_check()	void
controls_commander_elevation_check()	void
controls_turret_stab_check()	void
controls_gunner_palm_check)	void
controls_gunner_fast_slew_check()	void
controls_gunner_trigger_check()	void
controls_commander_palm_check()	void
controls_commander_fast_slew_check()	void
controls_commander_trigger_check()	void

Table 2.3-12: Turret Drive Routines.

Exit Routines:

These routines examine the gunner and commander traverse, elevation, stability, slew, and palm values, trigger values, values for the gunner and commander controls. If these controls are on, they are set to off or zero to properly and gracefully exit the controls module.

Name	Type of routine
controls_gunner_traverse_exit()	void
controls_gunner_elevation_exit()	void
controls_commander_traverse_exit()	void
controls_commander_elevation_exit()	void
controls_turret_stab_exit()	void
controls_gunner_palm_exit()	void
controls_gunner_fast_slew_exit()	void
controls_gunner_trigger_exit()	void
controls_commander_palm_exit()	void
controls_commander_fast_slew_exit()	void
controls_commander_trigger_exit()	void

Table 2.3-13: Exit Routines.

2.3.2.1.2 m2_ctl_mpc.c
(/simnet/release/src/vehicle/m1/src/m2_ctl_mpc.c [m2_ctl_mpc.c])

This CSU provides the master power controls interface for the M2 simulator. This module includes the following files:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"timers_dfn.h"
"timers.h"
"libidc.h"
"libidc_dfn.h"
"libmem.h"
"libmem_dfn.h"
"libnetwork.h"
"m2_main.h"
"m2_tmrs_df.h"
"m2_ctl_df.h"
"m2_driv_pn.h"
"m2_driv_mx.h"
"m2_turr_pn.h"
"m2_pots.h"
"m2_cntrl.h"
"m2_ramp.h"
"m2_elecsys.h"
"m2_fuelsys.h"
"m2_engine.h"

This file also defines the following parameters for use in controlling the master power mechanisms.

Variables	Type	Where typedef declared
real_steer_bar_val	real	sim_types.h
real_throttle_val	REAL	sim_types.h
real_accelerator_val	REAL	sim_types.h
hex_steer_bar_val	int	Standard
hex_throttle_val	int	Standard
hex_accelator_val	int	Standard
engine_accessory_val	char	Standard
tranmission_val	char	Standard
ramp_up_down_val	char	Standard
driver_panel_test_val	char	Standard
parking_brake status	char	Standard
cool_hi_temp_flash_count	int	Standard
cool_lo_level_flash_count	int	Standard
trans_oil_pres_lo_flash_count	int	Standard
fuel_filt_clog_flash_count	int	Standard
launcher_up_flash_count	int	Standard
air_clean_clog_flash_count	int	Standard
eng_oil_pres_lo_flash_count	int	Standard
cool_hi_temp_flash_event_edge	int	Standard
cool_lo_level_flash_event_edge	int	Standard
trans_oil_pres_lo_flash_event_edge	int	Standard
trans_oil_temp_hi_flash_event_edge	int	Standard
fuel_filt_clog_flash_event_edge	int	Standard
launcher_up_flash_event_edge	int	Standard
air_clean_clog_flash_event_edge	int	Standard
eng_oil_pres_lo_flash_event_edge	int	Standard
hull_radio_failure_status	int	Standard
hull_intercom_failure_status	int	Standard

Table 2.3-14: m2_ctl_mpc Variables.

This file defines the routines listed in the tables below.

Initialization Routines:

These routines initialize all starting values for the master power controls. If these controls are on, they are initialized to off or zero depending on the initial state.

Name	Type of routine
controls_steer_bar_init()	void
controls_throttle_init()	void
controls_engine_accessory_init()	void
controls_transmission_init()	void
controls_ramp_up_down_init()	void
controls_driver_panel_test_init()	void

Table 2.3-15: Initialization Routines.

Master Power Routines:

These routines check and evaluate the steer bar, engine accessories, throttle, ramp state, driver panel, cool hi and lo levels, tranmission, fuel filter, launcher state, air cleaner condition, and engine oil pressure and will return alarm messages to indicate that there may be a power control that needs adjustment or checking.

Name	Type of routine
controls_steer_bar_check()	void
controls_throttle_check()	void
controls_engine_accessory_check()	void
controls transmission_check()	void
controls_ramp_up_down_check()	void
controls_tone_cancel_check()	void
controls_driver_panel_test_check()	void
controls_cool_hi_temp_flash_check()	void
controls_cool_lo_level_flash_check()	void
controls_trans_oil_pres_lo_flash_check()	void
controls_trans_oil_temp_hi_flash_check()	void
controls_fuel_filt_clog_flash_check()	void
controls_launcher_up_flash_check()	void

Table 2.3-16: Master Power Routines.

Exit Routines:

These routines examine the steer bar,throttle, engine accessories, driver panel test module, and ramp state controls. If these controls are on, they are set to off or zero to properly exit the power controls module.

Name	Type of routine
controls_steer_bar_exit()	void
controls_throttle_exit()	void
controls_engine_accessory_exit()	void
controls_driver_panel_test_exit()	void
controls_ramp_up_down_exit()	void

Table 2.3-17: Exit Routines.

Miscellaneous Routines:

The **controls_driver_panel_test_on**() routine will turn or set to on the controls on the driver panel to specified starting values. If these values are incorrect or have somehow been changed to unacceptable values, **controls_driver_panel_test_restore**() will restore known good values to the driver panel status display.

Name	Type of routine
controls_driver_panel_test_on()	void
controls_driver_panel_test_restore()	void

Table 2.3-18: Miscellaneous Routines.

2.3.2.1.3 m2_ctl_tpc.c
(/simnet/release/src/vehicle/m1/src/m2_ctl_tpc.c [m2_ctl_tpc.c])

This CSU provides a controls interface for the M2 simulator.

This CSU provides the turret power controls interface for the M2 simulator. This module uses these include files:

stdio.h	m2_ctl_df.h
"sim_types.h"	"m2_cntrl.h"
"sim_dfns.h"	"m2_driv_pn.h"
"sim_macros.h"	"m2_turr_pn.h"
"timers_dfn.h"	"m2_driv_mx.h"
"timers.h"	"m2_turr_mx.h"
"libidc.h"	"m2_firectl.h"
"libidc_dfn.h"	"m2_handles.h"
"libmem.h"	"m2_weapons.h"
"libmem_dfn.h"	"m2_turret.h"
"libnetwork.h"	"m2_sound.h"
"libfail.h"	"m2_ammo.h"
"failure.h"	"m2_tmrs_df.h"
"m2_main.h"	"m2_electsys.h"

The file "m2_ctl_tpc" defines the following parameters for use in controlling the turret power mechanisms.

Variables	Type	Where typedef declared
turret_drive_val	char	Standard
commander_panel_tst_val	char	Standard
arm_safe+reset_val	char	Standard
gunner_launcher_val	char	Standard
sear_flashing_status	char	Standard
low_ammo_flashing_status	int	Standard
missile1_flashing_status	int	Standard
missile2_flashing_status	int	Standard
sear_flash_count	int	Standard
low_ammo_flash_count	int	Standard
missle1_flash_count	int	Standard
missile2_flash_count	int	Standard
sear_flash_event_edge	int	Standard
low_ammo_flash_event_edge	int	Standard
missile1_flash_event_edge	int	Standard
missile2_flash_event_edge	int	Standard
turret_ref_ind_status	nt	Standard
turret_radio_failure_status	int	Standard
turret_intercom_failure_status	int	Standard
gunner_turret_ref_translations	array of int	Standard
commander_turret_ref_translations	array of int	Standard

Table 2.3-19: m2_ctl_tpc Variables.

This file defines the routines listed in the table below.

Initialization Routines:

These routines initialize all starting values for the gunner and commander turret power controls. If these controls are on, they are initialized to off or zero depending on the initial state.

Name	Type of routine
controls_turret_drive_init()	void
controls_commander_panel_test_init()	void
controls_arm_safe_reset_init()	void
controls_gunner_launch_init()	void

Table 2.3-20: Initialization Routines.

Turret Power Routines:

The following routines control the turret power controls by checking the control state of these turret power gunner tow, missile select, low ammo override values, various power controls checks and the commander panel test check. If these values are out of acceptable range as determined by several other internal routines called by these routines, an error message will appear on the appropriate gunner or commander terminal screen.

Name	Type of Routine
controls_turret_drive_check()	void
controls_commander_panel_test_check()	void
controls_arm_safe_reset_check()	void
controls_low_ammo_override_check()	void
controls_gunner_launcher_check()	void
controls_misfire_check()	void
controls_gunner_tow_select_check()	void
controls_gunner_tow_test_check()	void
controls_gunner_missile1_check()	void
controls_gunner_missile2_check()	void
controls_ap_ss_check()	void
controls_he_ss_check()	void
controls_ap_lo_check()	void
controls_he_lo_check()	void
controls_ap_hi_check()	void
controls_he_hi_check()	void
controls_sear_flash_check()	void
controls_low_ammo_flash_check()	void
controls_missile1_flash_check()	void
controls_missile2_flash_check()	void

Table 2.3-21: Turret Power Routines.

Exit Routines:

These routines examine the gunner and commander turret power controls for the turret drive state, commander panel state, arming state, and launcher state. If these controls are on, they are set to off or zero to properly exit the controls module.

Name	Type of routine
controls_turret_drive_exit()	void
controls_commander_panel_test_exit()	void
controls_arm_safe_reset_exit()	void
controls_gunner_launcher_exit()	void

Table 2.3-22: Exit Routines.

Miscellaneous Routines:

These routines examine the commander panel test power controls state and if needed, restores the test panel values to the appropriate state.

Name	Type of routine
controls_commander_panel_test_on()	void
controls_commander_panel_test_restore()	void

Table 2.3-23: Miscellaneous Routines.

2.3.2.1.4 m2_ctl_hnp.c
(/simnet/release/src/vehicle/m1/src/m2_ctl_hnp.c [m2_ctl_hnp.c])

This CSU provides the hull no power controls interface for the M2 simulator. This module includes the following files:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libidc.h"
"libidc_dfn.h"
"libmem.h"
"libmem_dfn.h"
"failure.h"
"libfail.h"
"timers_dfn.h"
"m2_ctl_df.h"
"m2_cntrl.h"
"m2_tmrs_df.h"
"m2_driv_pn.h"
"m2_pots.h"
"m2_fuelsys.h"
"m2_idc.h"
"m2_driv_mx.h"

The file m2_ctl_hnp also defines the following parameters for use in controlling the hull no power mechanisms.

Variables	Type	Where typedef declared
parking_brake_val	char	Standard
real_service_val	REAL	sim_types.h
hex_service_brake_val	int	Standard
fuel_val	char	Standard
odometer_timer_number	int	Standard
hull_slope_ind_status	int	Standard
driver_slope_translations	int	Standard

Table 2.3-24: n2_ctl_hnp Variables.

This file defines the routines listed in the tables below.

Initialization Routines:

These routines initialize all starting values for the hull no power controls. If these controls are on, they are initialized to off or zero or a specified starting value depending on the initial state.

Name	Type of routine
controls_parking_brake_init()	void
controls_service_brake_init()	void
controls_fuel_init()	void

Table 2.3-25: Initialization Routines.

Hull No Power Routines:

These routines check and evaluate the the parking brake, service brake, odometer, and fuel level. They will return alarm messages to indicate that there may be a power control that needs adjustment or checking.

Name	Type of routine
controls_parking_brake_check()	void
controls_service_brake_check()	void
controls_odometer_check()	void
controls_fuel_check()	void

Table 2.3-26: Hull No Power Routines.

Exit Routines:

These routines examine the parking brake, service brake, and odometer value state. If these brake controls are off, they are set to on or zero to properly exit the controls module, and the odometer is properly reset to zero.

Name	Type of routine
controls_parking_brake_exit()	void
controls_service_brake_exit()	void
controls_odometer_exit()	void
controls_parking_brake_on()	extern void
controls_parking_brake_off()	extern void

Table 2.3-27: Exit Routines.

2.3.2.1.5 m2_ctl_tnp.c
(/simnet/release/src/vehicle/m1/src/m2_ctl_tnp.c [m2_ctl_tnp.c])

This CSU provides the turret no power controls interface for the M2 simulator. This module uses these include files:

"stdio.h"	"m2_ctl_df.h"
"sim_types.h"	"m2_cntrl.h"
"sim_dfns.h"	"m2_driv_pn.h"
"sim_macros.h"	"m2_turr_pn.h"
"mass_stdc.h"	"m2_tmrs_df.h"
"timers_dfn.h"	"m2_bcs.h"
"dtad.h"	"m2_cig.h"
"libidc.h"	"m2_pots.h"
"libidc_dfn.h"	"m2_dtrain.h"
"libmem.h"	"m2_isu.h"
"libmem_dfn.h"	"m2_turr_mx.h"
"libnetwork.h"	"m2_status.h"
"m2_idc.h"	"m2_vision.h"

This file defines the following parameters for use in controlling the turret no power mechanisms.

Variables	Type	Where typedef declared
real_cupola_val	REAL	sim_types.h
hex_cupola_val	int	Standard
turret_power_val	char	Standard
range_select_val	long	Standard
mag_select_val	int	Standard
gunner_brow_pad_val	char	Standard
commander_brow_pad_val	char	Standard
grid_azimuth_val	char	Standard
cupola_up_down_val	char	Standard
ammo_hei_can_hei_val	char	Standard
ammo_apds_can_hei_val	char	Standard
receive_flashing_status	char	Standard
send_flashing_status	char	Standard
internal_flashing_status	char	Standard
hei_flashing_status	char	Standard
apds_flashing_status	char	Standard
tow_flashing_status	char	Standard
dragon_flashing_status	char	Standard
receive_flash_count	int	Standard
send_flash_count	int	Standard
internal_flash_count	int	Standard
hei_flash_count	int	Standard
apds_flash_count	int	Standard
tow_flash_count	int	Standard
dragon_flash_count	int	Standard
receive_flash_event_edge	int	Standard
send_flash_event_edge	int	Standard

internal_fl;ash_event_edge	int	Standard
hei_flash_event_edge	int	Standard
apds_flash_event_edge	int	Standard
tow_flash_event_edge	int	Standard
dragon_flash_event_edge	int	Standard
gunner_slope_translations[SLOPE_NUM_SECTORS]	array of int	Standard
commander_slope_translations[SLOPE_NUM_SECTORS]	array int	Standard

Table 2.3-28: m2_ctl_tnp Variables.

This file defines the routines listed in the table below.

Initialization Routines:

These routines initialize all starting values for the gunner and commander brow pad, range select, magazine select, cupola up and down initial values, and ammo controls initial values. If these controls are on, they are initialized to off or zero depending on the initial state.

Name	Type of routine
controls_turret_power_init()	void
controls_cupola_init()	void
controls_range_select_init()	void
controls_mag_select_init()	void
controls_gunner_brow_pad_init()	void
controls_commander_brow_pad_init()	void
controls_cupola_up_down_init()	void
controls_controls_ammo_hei_can_init()	void
controls_ammo_apds_can_init()	void

Table 2.3-29: Initialization Routines.

Turret No Power Routines:

The following routines control the turret no power controls by checking the control state of these turret power vaues including the cupola, range select, magazine select, gunner and commander brow pad status, grid azimuth state, and ammo controls states. If these values are out of acceptable range as determined by other internal routines called by these routines, an error message will appear on the appropriate gunner or commander terminal screen.

Name	Type of routine
controls_turret_power_check()	void
controls_cupola_check()	void
controls_range_select_check()	void
controls_mag_select_check()	void
controls_gunner_brow_pad_check()	void
controls_commander_brow_pad_check()	void
controls_grid_azimuth_check()	void
controls_cupola_up_down_check()	void
controls_ammo_receive_check()	void
controls_ammo_send_check()	void
controls_ammo_internal_check()	void
controls_ammo_hei_check()	void
controls_ammo_apds_check()	void
controls_ammo_tow_check()	void
controls_ammo_dragon_check()	void
controls_ammo_hei_can_check()	void
controls_ammo_apds_can_check()	void
controls_receive_flash_check()	void
controls_send_flash_check()	void
controls_internal_flash_check()	void
controls_hei_flash_check()	void
controls_apds_flash_check()	void
control_tow_flash_check()	void
controls_dragon_flash_check()	void

Table 2.3-30: Turret No Power Routines.

Exit Routines:

These routines examine the gunner and commander brow pad controls status, cupola status, turret power status, and ammo controls status. If these controls are on, they are set to off or zero to properly exit the controls module.

Name	Type of routine
controls_turret_power_exit()	void
controls_gunner_brow_pad_exit()	void
controls_commander_brow_pad_exit()	void
controls_cupola_up_down_exit()	void
controls_ammo_hei_can_exit()	void
controls_ammo_apds_can_exit()	void

Table 2.3-31: Exit Routines.

2.3.2.2 Finite State Machines

In general, finite state machines describe the appropriate actions to take when a control changes value. The following are examples of the type of functions that are contained in the files listed above:

i) When the driver of an m2 tank presses the accelerator, the simulation must first discern the states of the engine, transmission, emergency brake, etc. before the actions' effect can be determined

ii) When the gunner on an m1 tank pulls the trigger, the simulation must first check the states of the gunners palm grip, commanders palm grip, the weapon loaded, etc. before determining the necessary actions resulting from the trigger pull.

The following CSU's perform these functions:

m2_ctl_fsm.c
m2_handles.c
m2_firectl.c

2.3.2.2.1 m2_ctl_fsm.c
(./simnet/release/src/vehicle/m2/src/m2_ctl_fsm.c [m2_ctl_fsm.c)

This CSU provides the finite power state controls interface for the M2 simulator. This module includes the following files:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"timers.h"
"libidc.h"
"libidc_dfn.h"
"libmem.h"
"libmem_dfn.h"
"m2_main.h"
"m2_ctl_df.h"
"m2_driv_pn.h"
"m2_turr_pn.h"
"m2_pots.h"
"m2_cntrl.h"
"m2_bcs.h"
"m2_isu.h"
"libnetwork.h"

This file also defines the following parameters for use in controlling the finite power state mechanisms.

Variables	Type	Where typedef declared
controls_hull_status_on_edge	int	Standard
controls_hull_status_off_edge	int	Standard
controls_hull_electsys_val	int	Standard
controls_hull_electsys_off_edge	char	Standard
controls_hull_electsys_on_edge	char	Standard
controls_turret_backup_electsys_val	char	Standard
controls_turret_backup_electsys_off_edge	char	Standard
controls_turret-backup_electsys_on_edge	char	Standard
controls_failure_val	int	Standard
controls_failure_off_edge	int	Standard
controls_failure_on_edge	int	Standard
controls_turret_drive_val	int	Standard
controls_turret_drive_val	int	Standard
controls_turret_drive_on_edge	int	Standard
controls_turret_driveoff_edge	int	Standard
controls_turret_power_systeem_val	int	Standard
controls_turret_power_system_on_edge	int	Standard
controls_turret_power_system_edge	int	Standard
controls_master_power_val	char	Standard

Table 2.3-32: m2_ctl_fsm Variables.

This file defines the routines listed in the tables below.

Hull State Transition Routines

These routines check the next state or value for the hull and master power levels. If a no power state is detected then the hull status, hull electrical system values are set to specified initial values which are usually defined as an OFF state. They will also test for disallowed values and print an error message saying an impossible state has been entered, and will reset that state with the routine **controls_fsm_init()**.

Name	Type of routine
controls_hull_no_power_next_state()	void
controls_master_power_next_state()	void

Table 2.3-33: Hull State Transition Routines.

Turret State Transition Routines

These routines check the next state or value for the turret electrical system and turret drive power state. If a no power state is detected then the turret electrical system and turret drive values are set to specified initial values which are usually defined as an OFF state. They will also test for disallowed values and print an error message saying an impossible state has been entered, and will reset that state with the routine **controls_fsm_init**().

Name	Type of routine
controls_turret_no_power_next_state()	void
controls_master_power_next_state()	void
controls_turret_drive_next_state()	void
controls_lamp_init ()	void

Table 2.3-34: Turret State Transition Routines.

2.3.2.2.2 m2_handles.c
(./simnet/release/src/vehicle/m2/src/m2_handles.c [m2_handles.c)

The CSU m2_handles.c contains state machines that keep track of the gunners and commanders palm grips, and the associated control of the respective turrets. This is the gunner's and commander's controls module. Information is passed between the controls module and the parts of the simulation that need the information (turret, gunnery, and laser range finder systems). The file checks to see whether the handles are working, and which set of handles is engaged.

Includes:

"sim_types.h"
"sim_dfns.h"
"sim_macros.h:
"failure.h"
"libfail.h"
"m2_bcs.h"
"m2_weapons.h"
"m2_turret.h"
"m2_launcher.h"

Defines:

Launcher Constants	Value
LAUNCHER_UP_VAL	0
LAUNCHER_DOWN_VAL	1
LAUNCHER_IDLE_VAL	2

Procedure Declarations:

handles_launcher_check()
handles_launcher_default()

int variable declarations:

gunner_control_status	-- maintenance status: normally WORKING
cmdr_control_status	
gunner_control_engaged	-- booleans which tell whether palm switches are engaged
cmdr_control_engaged	
gunner_fast_trav_pushed	-- TRUE or FALSE
cmdr_fast_trav_pushed	-- Level triggered
gunner_trigger_pushed	-- Either TRUE or FALSE
cmdr_trigger_pushed	-- Level triggered
gunner_trigger_edge	
cmdr_trigger_edge	
trigger_is_pulled	
launcher_control	
old_launcher_control	

REAL variable declarations -- Normalized values from -1.0 to +1.0 which are received directly from the controls module and passed to the turret if the appropriate palm switch is engaged.

gunner_traverse
cmdr_traverse

gunner_elevate
cmdr_elevate

2.3.2.2.2.1 handles_simul

This routine simulates the commander's controls on a tick by tick basis. It checks to see if handles are working and which handles are engaged.

Calls	
Function	**Where Described**
turret_handles_values	Section 2.3.6.1.1.6
weapons_trigger_is_pulled	Section 2.3.3.2.12
weapons_trigger_is_released	Section 2.3.3.2.8
handles_launcher_check	Section 2.3.2.2.2.25
handles_launcher_default	Section 2.3.2.2.2.26
weapons_fire	Section 2.3.3.2.4

Table 2.3-35: handles_simul Information.

2.3.2.2.2.2 handles_gunner_control_fixed

This routine repairs the gunner's control.

2.3.2.2.2.3 handles_gunner_control_broken

This routine causes the gunner's control to fail.

2.3.2.2.2.4 handles_commander_control_fixed

This routine repairs the commander's control.

2.3.2.2.2.5 handles_commander_control_broken

This routine causes the gunner's control to fail.

2.3.2.2.2.6 handles_gunner_palm_on

This routine engages the gunner's control.

2.3.2.2.2.7 handles_gunner_palm_off

This routine disengages the gunner's palm switch. It is called by the controls module when the gunner's palm switch is released, or when turret power is turned off while the gunner's palm switch is engaged.

2.3.2.2.2.8 handles_commander_palm_on

This routine engages the commander's control.

2.3.2.2.2.9 handles_commander_palm_off

This routine disengages the commander's palm switch. It is called by the controls module when the commander's palm switch is released, or when turret power is turned off while the commander's palm switch is engaged.

2.3.2.2.2.10 handles_gunner_fast_slew_on

This routine sets *gunner_fast_trav_pushed* equal to TRUE. The gunner's palm switch is set to fast traverse speed.

2.3.2.2.2.11 handles_gunner_fast_slew_off

This routine sets *gunner_fast_trav_pushed* equal to FALSE. The gunner's palm switch is set to normal traverse speed.

2.3.2.2.2.12 handles_commander_fast_slew_on

This routine sets *cmdr_fast_trav_pushed* equal to TRUE. The commander's palm switch is set to fast traverse speed.

2.3.2.2.2.13 handles_commander_fast_slew_off

This routine sets *cmdr_fast_trav_pushed* equal to FALSE. The commander's palm switch is set to normal traverse speed.

2.3.2.2.2.14 handles_set_gunner_elevation

This routine sets the gunner's elevation rate to *elevation_rate*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
elevation_rate	register REAL	sim_types.h

Table 2.3-36: handles_set_gunner_elevation Information.

2.3.2.2.2.15 handles_set_commander_elevation

This routine sets the commander's elevation rate to *elevation_rate*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
elevation_rate	register REAL	sim_types.h

Table 2.3-37: handles_set_commander_elevation Information.

2.3.2.2.2.16 handles_set_gunner_traverse

This routine sets the gunner's traverse rate to *traverse_rate*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
traverse_rate	register REAL	sim_types.h

Table 2.3-38: handles_set_gunner_traverse Information.

2.3.2.2.2.17 handles_set_commander_traverse

This routine the commander's traverse rate to *traverse_rate*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
traverse_rate	register REAL	sim_types.h

Table 2.3-39: handles_set_commander_traverse Information.

2.3.2.2.2.18 handles_gunner_trigger_depressed

This routine sets gunner_trigger_pushed and gunner_trigger_edge equal to TRUE when the gunner's trigger is depressed.

2.3.2.2.2.19 handles_commander_trigger_depressed

This routine sets cmdr_trigger_pushed and cmdr_trigger_edge equal to TRUE when the commander's trigger is depressed.

2.3.2.2.2.20 handles_gunner_trigger_released

This routine sets *gunner_trigger_pushed* equal to FALSE when the gunner's trigger is released.

2.3.2.2.2.21 handles_commander_trigger_released

This routine sets *cmdr_trigger_pushed* equal to FALSE when the commander's trigger is released.

2.3.2.2.2.22 handles_launcher_up

This routine raises the launcher.

2.3.2.2.2.23 handles_launcher_down

This routine lowers the launcher.

2.3.2.2.2.24 handles_launcher_idle

This routine sets launcher_control equal to LAUNCHER_IDLE_VAL.

2.3.2.2.2.25 handles_launcher_check

This routine is called on a tick by tick basis in order to set the launcher position.

Calls	
Function	**Where Described**
launcher_up	Section 2.3.6.1.4.3
launcher_down	Section 2.3.6.1.4.4
launcher_idle	Section 2.3.6.1.4.5

Table 2.3-40: handles_launcher_check Information.

2.3.2.2.2.26 handles_launcher_default

This routine sets the launcher to the idle mode.

Calls	
Function	**Where Described**
launcher_idle	Section 2.3.6.1.4.5

Table 2.3-41: handles_launcher_default Information.

2.3.2.2.2.27 handles_init

This routine initializes the handles.

Calls	
Function	**Where Described**
fail_init_failure	Section 2.5.4.11.2

Table 2.3-42: handles_init Information.

2.3.2.2.3 m2_firectl.c
(./simnet/release/src/vehicle/m2/src/m2_firectl.c [m2_firectl.c])

The file m2_firectl.c computes the states important to firing weapons.

Includes:
- "stdio.h"
- "sim_types.h"
- "sim_dfns.h"
- "sim_macros.h"
- "basic.h"
- "m2_turr_mx.h"
- "m2_ctl_df.h"
- "m2_cntrl.h"
- "m2_firectl.h"
- "m2_laser.h"
- "m2_ammo_df.h"
- "m2_ammo.h"

Defines:

Symbol	Value
WEAPON_DELAY	5

int variable declarations:
- mag_select_val
- arm_safe_reset_val
- weapon_counter

2.3.2.2.3.1 firectl_init

This routine initializes the fire control system variables *mag_select_val*, *arm_safe_reset_val*, and *weapon_counter*.

2.3.2.2.3.2 firectl_simul

This routine provides the tick by tick simulation of the fire control system. The weapon counter is decremented.

2.3.2.2.3.3 firectl_gps_mag_4x

This routine sets the gunner's primary sight magnification to 4X.

2.3.2.2.3.4 firectl_gps_mag_12x

This routine sets the gunner's primary sight magnification to 12X.

2.3.2.2.3.5 firectl_gps_mag_status

This routine returns the magnification of the gunner's primary sight.

Return Values		
Return Value	**Type**	**Meaning**
mag_select_val	int	The gps magnification selection

Table 2.3-43: firectl_gps_mag_status Information.

2.3.2.2.3.6 firectl_arm

This routine sets the fire control system state to armed.

Calls	
Function	**Where Described**
controls_firectl_arm_on	Section 2.3.2

Table 2.3-44: firectl_arm Information.

2.3.2.2.3.7 firectl_safe

This routine sets the fire control system state to safe.

Calls	
Function	**Where Described**
controls_firectl_arm_off	Section 2.3.2

Table 2.3-45: firectl_safe Information.

2.3.2.2.3.8 firectl_arm_safe_reset_status

This routine returns the status of the fire control system.

Return Values		
Return Value	**Type**	**Meaning**
arm_safe_reset_val	int	The fire control system status

Table 2.3-46: firectl_arm_safe_reset_status Information.

2.3.2.2.3.9 firectl_weapon_removed

This routine sets weapon_counter equal to WEAPON_DELAY.

2.3.2.2.3.10 firectl_weapon_ready

This routine returns TRUE if the weapon is ready to fire (the weapon counter equals zero), and FALSE otherwise.

Return Values		
Return Value	**Type**	**Meaning**
weapon_counter == 0	int	If TRUE, the weapon counter is equal to zero; If FALSE, the weapon counter is not equal to zero.

Table 2.3-47: firectl_weapon_ready Information.

2.3.2.2.3.11 firectl_25mm_ready_to_fire

This routine determines if the 25mm weapon is ready to fire.

Return Values		
Return Value	**Type**	**Meaning**
(controls_turret_power_status () != CONTROLS_STATE_ TURRET_NO_POWER) && (controls_turret_drive_system _status() == ON) && (arm_safe_reset_val == TC_ ARM_VAL) && (!electsys_drive_malfunction_ status()) && (!electsys_25mm_gun_ malfunction_status())	int	If TRUE, the weapon is ready to fire; If FALSE, the weapon is not ready to fire.

Calls	
Function	**Where Described**
controls_turret_power_status	Section 2.3.2
controls_turret_drive_system _status	Section 2.3.2
electsys_drive_malfunction_s tatus	Section 2.3.6.3.1.12
electsys_25mm_gun_malfunc tion_status	Section 2.3.6.3.1.14

Table 2.3-48: firectl_25mm_ready_to_fire Information.

2.3.2.2.3.12 firectl_tow_ready_to_fire

This routine determines if the tow missile is ready to fire. If the following conditions are met, the tow missile is ready to fire:
1. Turret power is on.
2. The turret drive system is on.
3. The gunner's primary sight magnification is 12X
4. The tow missile is armed.
5. There is no electrical system malfunction.

Return Values		
Return Value	**Type**	**Meaning**
(controls_turret_power_status () != CONTROLS_STATE_ TURRET_NO_POWER) && (controls_turret_drive_system _status() == ON) && (arm_safe_reset_val == TC_ ARM_VAL) && (!electsys_drive_malfunction_ status()) && (!electsys_tow_circuit_open_ status()) && (mag_select_val = GN_12X_VAL)	int	If TRUE, the tow missile is ready to fire; If FALSE, the tow missile is not ready to fire.

Calls	
Function	**Where Described**
controls_turret_power_status	Section 2.3.2
controls_turret_drive_system _status	Section 2.3.2
electsys_drive_malfunction_s tatus	Section 2.3.6.3.1.12
electsys_tow_circuit_open_st atus	Section 2.3.6.3.1.16

Table 2.3-49: firectl_tow_ready_to_fire Information.

2.3.2.3 Specialized Output Devices

As stated in section 2.1, the output devices for a simulation are often vehicle specific. Values are computed which need to be displayed to the operator, and new values are sent to the low level control routines (see 2.1.4). This is accomplished with the following CSU's:

m2_alpha.c
m2_gages.c
m2_meter.c
m2_odom.c
m2_pots.c
m2_slope.c

2.3.2.3.1 m2_alpha.c

(./simnet/release/src/vehicle/m2/src/m2_alpha.c [m2_alpha.c])

This CSU models output devices on the M2. This is the source file for alpha_numeric display.

Includes:
"stdio.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"fifo_dfn.h"
"fifo.h"
"m2_alpha.h"
"m2_mem_dfn.h"
"m2_alpha_df.h"
"m2_ctl_df.h"

Defines:
dig_to_ascii

char declarations:
alpha_char
mils_string [MILS_STRING_SIZE]
load_string [LOAD_STRING_SIZE]

int variable declarations:
load
old_load
mils
old_mils

Procedure declarations:
alpha_send_mils()
alpha_send_load()

2.3.2.3.1.1 alpha_init

This routine initializes the alpha-numeric displays.

Calls	
Function	**Where Described**
fifo_init	Section 2.6.8.3.1
alpha_reset	Section 2.3.2.3.1.2

Table 2.3-50: alpha_init Information.

2.3.2.3.1.2 alpha_reset

This routine resets the alpha-numeric display.

Calls	
Function	**Where Described**
fifo_enqueue	Section 2.6.8.2.1

Table 2.3-51: alpha_reset Information.

2.3.2.3.1.3 alpha_send_mils

Given *radians*, this routine sends mils to the alpha-numeric display.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
radians	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
units	int	Standard
tens	int	Standard
hundreds	int	Standard
thousands	int	Standard
remainder	int	Standard
Calls		
Function	**Where Described**	
dig_to_ascii	Section 2.3.2.3.1	
fifo_enqueue	Section 2.6.8.2.1	

Table 2.3-52: alpha_send_mils Information.

2.3.2.3.1.4 alpha_send_load

Given *radians*, this routine sends the load to the alpha-numeric display.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
radians	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
degrees	REAL	sim_types.h
shift_degrees	REAL	sim_types.h
Calls		
Function	**Where Described**	
rad_to_deg	sim_macros.h	
fifo_enqueue	Section 2.6.8.2.1	

Table 2.3-53: alpha_send_load Information.

2.3.2.3.1.5 alpha_send

Given *radians*, this routine sends mils and load to the alpha-numeric display.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
radians	REAL	sim_types.h
Calls		
Function	**Where Described**	
alpha_send_mils	Section 2.3.2.3.1.3	
alpha_send_load	Section 2.3.2.3.1.4	

Table 2.3-54: alpha_send Information.

2.3.2.3.1.6 alpha_get_load

This routine returns load.

Return Values		
Return Value	**Type**	**Meaning**
load	int	The load

Table 2.3-55: alpha_get_load Information.

2.3.2.3.2 m2_gages.c

(./simnet/release/src/vehicle/m2/src/m2_gages.c [m2_gages.c])

This CSU models gauges on the M2.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"m2_cons.h"
"m2_gages.h"

Defines:

Symbol	Valuc
IDLE_SPEED	820
MAX_SPEED	2700
SCALE_INTERCEPT	0.436
OIL_CUBIC	-.00000044755
OIL_QUAD	.00052867
OIL_LIN	-.231049
OIL_CONST	38.888
TORQUE_TEMP_FACTOR	.0000004
BLOCK_TEMP_FACTOR	.00001
RADIATOR_TEMP_FACTOR	.00001
TORQUE_TEMP_CONST	2500

int variable declarations and initialization:

counter = 0

REAL declarations:

scale
oil_press
oil_temp
coolant_temp

2.3.2.3.2.1 gage_oil_pressure

This routine calculates the oil pressure.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
speed	REAL	sim_types.h
o_temp	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
oil_press	REAL	the oil pressure
Calls		
Function	**Where Described**	
abs	sim_macros.h & abs.h	

Table 2.3-56: gage_oil_pressure Information.

2.3.2.3.2.2 gage_oil_temperature

This routine calculates the oil temperature.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
old_temp	REAL	sim_types.h
cool_temp	REAL	sim_types.h
speed	REAL	sim_types.h
torque	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
oil_temp	REAL	the oil temperature

Table 2.3-57: gage_oil_temperature Information.

2.3.2.3.2.3 gage_coolant_temperature

This routine calculates the coolant temperature.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
old_temp	REAL	sim_types.h
o_temp	REAL	sim_types.h
speed	REAL	sim_types.h
fail_factor	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
coolant_temp	REAL	The coolant temperature

Table 2.3-58: gage_coolant_temperature Information.

2.3.2.3.3 m2_meter.c

(./simnet/release/src/vehicle/m2/src/m2_meter.c [m2_meter.c])

Meters on the M2 are modelled by this CSU.

Includes:
"sim_types.h"
"sim_dfns.h"
"m2_mtr_df.h"
"m2_ctl_df.h"
"m2_fuel_df.h"
"m2_driv_pn.h"
"m2_meter.h"
"m2_cntrl.h"

char declarations:
speed_set_val
fuel_set_val
volt_set_val
temp_set_val
press_set_val

REAL variable declarations and initialization:
bottom_fuel_full = BOTTOM_FUEL_FULL

2.3.2.3.3.1 meter_init

This routine initializes the analog meter outputs, speedometer, fuel guage, voltmeter, temperature guage, and pressure guage.

2.3.2.3.3.2 meter_speed_set

This routine sets the reading on the speedometer to *val*, the input value.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp1	char	Standard
temp2	int	Standard
Calls		
Function	**Where Described**	
controls_hull_power_status	Section 2.3.2	
controls_failure_status	Section 2.3.2	
idc_output_set	Section 2.3.2	

Table 2.3-59: meter_speed_set Information.

2.3.2.3.3.3 meter_fuel_set

This routine sets the value on the fuel guage based on the value of *val*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp1	char	Standard
temp2	int	Standard
Calls		
Function	**Where Described**	
controls_failure_status	Section 2.3.2	
idc_output_set	Section 2.3.2	

Table 2.3-60: meter_fuel_set Information.

2.3.2.3.3.4 meter_volt_set

This routine sets the reading on the voltmeter based on the value of *val*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp1	char	Standard
temp2	int	Standard
Calls		
Function	**Where Described**	
controls_hull_power_status	Section 2.3.2	
controls_failure_status	Section 2.3.2	
idc_output_set	Section 2.3.2	

Table 2.3-61: meter_volt_set Information.

2.3.2.3.3.5 meter_temp_set

This routine sets the reading on the driver's temperature guage based on the value of *val*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp1	char	Standard
temp2	int	Standard
Calls		
Function	**Where Described**	
controls_hull_power_status	Section 2.3.2	
controls_failure_status	Section 2.3.2	
idc_output_set	Section 2.3.2	

Table 2.3-62: meter_temp_set Information.

2.3.2.3.3.6 meter_press_set

This routine sets the reading on the driver's oil pressure guage based on the value of *val*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
press1	char	Standard
press2	int	Standard
Calls		
Function	**Where Described**	
controls_hull_power_status	Section 2.3.2	
controls_failure_status	Section 2.3.2	
idc_output_set	Section 2.3.2	

Table 2.3-63: meter_press_set Information.

2.3.2.3.4 m2_odom.c

(./simnet/release/src/vehicle/m2/src/m2_odom.c [m2_odom.c])

Odometers on the M2 are modelled by this CSU.

Includes:
"stdio.h"
"sim_types.h"
"sim_macros.h"
"sim_dfns.h"
"m2_cons.h"
"m2_dtrain.h"
"libfail.h"

REAL variable declarations:
elapsed_km
elapsed_miles

int variable declarations:
elapsed_km_int
km_recorded
elapsed_miles_int
miles_recorded
mile_counter

2.3.2.3.4.1 odometer_init

This routine initializes the odometer by setting all values to zero.

2.3.2.3.4.2 odometer_simul

This routine does a tick-by-tick simulation of the odometer. It uses drivetrain speed to update speed, elapsed miles, and elapsed km.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
speed	REAL	sim_types.h

Calls	
Function	**Where Described**
drivetrain_get_vehicle_speed	Section 2.3.6.2.4.8
abs	sim_macros.h & abs.h
controls_odometer_pulse	Section 2.3.2
sfail_event_occurred	Section 2.5.4.21.1

Table 2.3-64: odometer_simul Information.

2.3.2.3.4.3 odom_set_initial_distance_km

This routine sets the the initial elapsed km to *distance*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
distance	REAL	sim_types.h

Table 2.3-65: odom_set_initial_distance_km Information.

2.3.2.3.4.4 vehicle_get_elapsed_km

This routine returns elapsed kilometers.

Return Values		
Return Value	**Type**	**Meaning**
elapsed_km	REAL	The elapsed kilometers

Table 2.3-66: vehicle_get_elapsed_km Information.

2.3.2.3.4.5 odometer_get_elapsed_km

This routine returns elapsed kilometers.

Return Values		
Return Value	Type	Meaning
elapsed_km	REAL	The elapsed kilometers

Table 2.3-67: odometer_get_elapsed_km Information.

2.3.2.3.4.6 odometer_get_elapsed_miles

This routine returns elapsed miles.

Return Values		
Return Value	Type	Meaning
elapsed_miles	REAL	The elapsed miles

Table 2.3-68: odometer_get_elapsed_miles Information.

2.3.2.3.4.7 odometer_mile_counter_reset

This routine sets the mile counter equal to the miles recorded.

2.3.2.3.3.8 odometer_mile_counter

This routine returns the trip mileage.

Internal Variables		
Internal Variable	Type	Where Typedef Declared
mile_difference	int	Standard
Return Values		
Return Value	Type	Meaning
mile_difference	int	the trip mileage

Table 2.3-69: odometer_mile_counter Information.

2.3.2.3.5 m2_pots.c
(./simnet/release/src/vehicle/m2/src/m2_pots.c [m2_pots.c])

Potentiometers on the M2 are modelled by this CSU.

This file contains the procedures relating to the potentiometers, which translate hex potentiometer values between 00 and FF into real numbers and call the appropriate M2 subsystems with those real values.

Includes:
"stdio.h"
"simstdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libidc_dfn.h"
"libnetwork.h"
"m2_pots_df.h"
"m2_cali_df.h"
"libpots.h"

The following variables represent the calibrated hex values corresponding to the pot extremes, i.e., cm_tur_trav_l is the hex value of the commanders turret traverse handle when it is at the full left position. These values are used to convert hex pot values to real numbers.

The commander's turret traverse: left, right, centered:
tc_tur_trav_l
tc_tur_trav_r
tc_tur_trav_c

The commander's turret elevate: depress, raise, centered:
tc_tur_elev_d
tc_tur_elev_r
tc_tur_elev_c

The gunner's turret traverse: left, right, centered:
gn_tur_trav_l
gn_tur_trav_r
gn_tur_trav_c

The gunner's turret elevate: depress, raise, centered:
gn_tur_elev_d
gn_tur_elev_r
gn_tur_elev_c

The driver's steering bar: left, right, centered:
dr_steer_l
dr_steer_r
dr_steer_c

The driver's throttle: zero, full:
dr_throttle_z
dr_throttle_f

The driver's service brake: zero, full:
dr_s_brake_z
dr_s_brake_f

The driver's accelerator: zero, full:
dr_accelerator_z
dr_accelerator_f

2.3.2.3.5.1 pots_init

This routine initializes the potentiometers.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
pots	pointer to FILE	
pots_error	int	Standard
line	int	Standard
tmp_str[40]	char	Standard
Errors		
Error Name	**Reason for Error**	
POTS: can't open calibrate file	A call to FOPEN returned NULL.	
POTS: can't continue because of calibration error	pots_error == TRUE	
POTS: can't close calibrate file	A call to FCLOSE returned EOF.	
Calls		
Function	**Where Described**	
nprintf	Section 2.1.1.3.1.34.1	
pots_check_two	Section 2.1.4.1.2.6.1	
pots_check_three	Section 2.1.4.1.2.5.1	

Table 2.3-70: pots_init Information.

2.3.2.3.5.2 pots_comm_trav_real

Given the potentiometer value *pot*, this routine returns a scaled real value between -1.0 and +1.0 for the commander's turret traverse control.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pot	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
pots_scale_lcr(...)	REAL	The scaled value of the commander's turret traverse control.
Calls		
Function	**Where Described**	
pots_scale_lcr	Section 2.1.4.1.2.2.1	

Table 2.3-71: pots_comm_trav_real Information.

2.3.2.3.5.3 pots_comm_elev_real

Given the potentiometer value *pot*, this routine returns a scaled real value between -1.0 and +1.0 for the commander's turret elevation control.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pot	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
pots_scale_lcr(...)	REAL	The scaled value of the commander's elevation control
Calls		
Function	**Where Described**	
pots_scale_lcr	Section 2.1.4.1.2.2.1	

Table 2.3-72: pots_comm_elev_real Information.

2.3.2.3.5.4 pots_cupola_real

Given the potentiometer value *pot*, this routine returns a scaled real value between -1.0 and +1.0 for the cupola control.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pot	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
pots_scale_lr_both(...)	REAL	The scaled value of the cupola control
Calls		
Function	**Where Described**	
pots_scale_lr_both	Section 2.1.4.1.2.3.1	

Table 2.3-73: pots_cupola_real Information.

2.3.2.3.5.5 pots_gunn_trav_real

Given the potentiometer value *pot*, this routine returns a scaled real value between -1.0 and +1.0 for the gunner's turret traverse control.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pot	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
pots_scale_lcr(...)	REAL	The scaled value of the gunner's turret traverse control
Calls		
Function	**Where Described**	
pots_scale_lcr	Section 2.1.4.1.2.2.1	

Table 2.3-74: pots_gunn_trav_real Information.

2.3.2.3.5.6 pots_gunn_elev_real

Given the potentiometer value *pot*, this routine returns a scaled real value between -1.0 and +1.0 for the gunner's turret elevation control.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pot	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
pots_scale_lcr(...)	REAL	The scaled value of the gunner's elevation control
Calls		
Function	**Where Described**	
pots_scale_lcr	Section 2.1.4.1.2.2.1	

Table 2.3-75: pots_gunn_elev_real Information.

2.3.2.3.5.7 pots_steer_bar_real

Given the potentiometer value *pot*, this routine returns a scaled real value between -1.0 and +1.0 for the driver's steering bar.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pot	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
pots_scale_lcr(...)	REAL	The scaled value of the driver's steering bar
Calls		
Function	**Where Described**	
pots_scale_lcr	Section 2.1.4.1.2.2.1	

Table 2.3-76: pots_steer_bar_real Information.

2.3.2.3.5.8 pots_throttle_real

Given the potentiometer value *pot*, this routine returns a scaled real value between -1.0 and +1.0 for the driver's throttle.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pot	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
pots_scale_lr_pos(...)	REAL	The scaled value of the driver's throttle
Calls		
Function	**Where Described**	
pots_scale_lr_pos	Section 2.1.4.1.2.4.1	

Table 2.3-77: pots_throttle_real Information.

2.3.2.3.5.9 pots_service_brake_real

Given the potentiometer value *pot*, this routine returns a scaled real value between -1.0 and +1.0 for the driver's service brake.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pot	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
pots_scale_lr_pos(...)	REAL	The scaled value of the driver's service brake.
Calls		
Function	**Where Described**	
pots_scale_lr_pos	Section 2.1.4.1.2.4.1	

Table 2.3-78: pots_service_brake_real Information.

2.3.2.3.5.10 pots_accelerator_real

Given the potentiometer value *pot*, this routine returns a scaled real value between -1.0 and +1.0 for the driver's accelerator.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pot	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
pots_scale_lr_pos(...)	REAL	The scaled value of the driver's accelerator
Calls		
Function	**Where Described**	
pots_scale_lr_pos	Section 2.1.4.1.2.4.1	

Table 2.3-79: pots_accelerator_real Information.

2.3.2.3.6 m2_slope.c

(./simnet/release/src/vehicle/m2/src/m2_slope.c [m2_slope.c])

This CSU models output devices on the M2.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"libturret.h"
"libmatrix.h"
"bigwheel.h"
"libkin.h"
"libhull.h"
"timers.h"
"timers_dfn.h"
"m2_turret.h"
"m2_cntrl.h"

Defines:

Symbol	Value
SLOPE_CENTERED_DEGREES	(10.0)
COS_SLOPE_CENTERED	(0.98480775301)

REAL declarations:

cos_hull_slope

2.3.2.3.6.1 slope_simul

This routine determines what the slope bubble for the hull and turret should be, and passes the information to the controls module.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
hull_dir	REAL	sim_types.h
turret_dir	REAL	sim_types.h
slope_centered	BOOLEAN	sim_types.h

Calls	
Function	**Where Described**
kinematics_get_slope_ind	Section 2.5.8.1.8
turret_get_ref_ind	Section 2.5.5.2.16
controls_hull_slope_ind	Section 2.3.2
controls_turret_slope_ind	Section 2.3.2

Table 2.3-80: slope_simul Information.

2.3.2.3.6.2 slope_get_cos_hull_slope

This routine returns the cosine of the slope bubble.

Return Values		
Return Value	**Type**	**Meaning**
cos_hull_slope	REAL	The cosine of the hull slope bubble

Table 2.3-81: slope_get_cos_hull_slope Information.

2.3.3 M2 Weapons

The M2 fires unguided, or ballistic, rounds as well as TOW missiles. The simulation must model the processes associated with the weapons systems which take place within the vehicle. It also must model the flyout of the round or missile and the reaction to impacts.

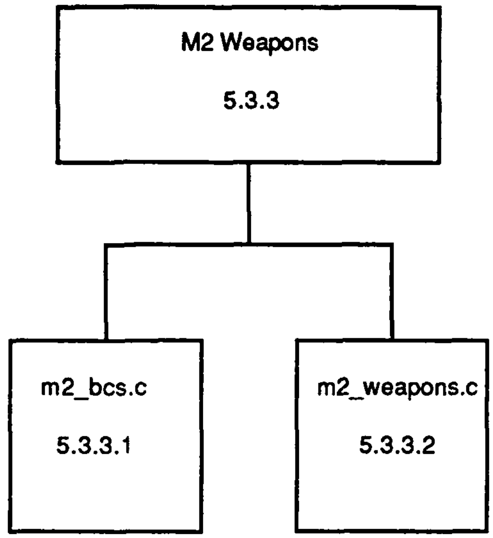

Figure 2.3-4: Structure of the M2 Weapons CSC.

The M2 accounts for the ballistic trajectory of the rounds fired in ballistic computer systems. These systems take range to target and turret traverse rate information coupled with knowledge about the round to be fired and calculate superelevation and lead angles.

Calculation of the superelevation and lead angles takes place in m2_bcs.c. The range used in the M2 is manually set by the user. This system is modeled in m2_bcs.c. This routine tells the CIG the current range so the range can be displayed.

Knowledge of the round to be fired is also maintained in m2_bcs.c. The required superelevation and time of flight for a particular round at a particular range is found using routines in libball. This information is used to calculate lead angle.

Commands to fire a ballistic round are initiated in the controls code. There are a number of steps to be performed when a round is fired and these tasks are performed m2_weapons.c. The availability of the round is checked by polling the munitions management code. If a round is available, the direction of the gun tube is determined. This is a combination of the gunner's line of sight, the superelevation and lead angle determined in the ballistics computer and a random offset to model dispersion. The dispersion is found using routines in the math library. Libball is called to actually fire the round. If the round is successfully fired, the sound of firing is made, the ammunition is decremented and a message is put onto the net.

Certain vehicle specific functions associated with firing ballistic rounds are performed through m2_weapons.c. The gun on the M2 fires repetitively and the task of tracking the rates of fire and firing the rounds is also performed in m2_weapons.c.

The M2 fires a TOW missile. Firing the missile is similar to firing a ballistic round. The controls code calls m2_weapons.c when the trigger is pulled. The weapons code times the duration of the trigger pull and checks a number of conditions. Items of concern are vehicle speed, TOW launcher up, and missile selected. If all the necessary conditions are met for the required amount of time and a missile is available, an attempt to fire a missile is made. The attempt is made by calling a routine in libmissile. If the attempt is successful, libmissile puts a message on the network and m2_weapons.c generates the appropriate sound.

The flyout of the TOW is modeled in libmissile. M2_weapons.c sends missile control signals to libmissile. Libmissile uses this information along with an understanding the the missile characteristics to generate a series of missile trajectory chords. This chord information is sent to the CIG and to the network. The CIG code determines when an impact occurs and calls libmissile. Libmissile uses the same routines used for ballistic rounds to generate the necessary effects and place messages on the network.

The following two CSU's contain the M2 specific routines necessary for this simulation.

m2_bcs.c
m2_weapons.c

2.3.3.1 m2_bcs.c
(./simnet/release/src/vehicle/m2/src/m2_bcs.c [m2_bcs.c])

The M2's ballistic weapons are modelled by this CSU.

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mun_type.h"
"timers.h"
"timers_dfn.h"
"libmatrix.h"
"libball.h"
"m2_bcs.h"
"m2_cntrl.h"
"m2_turret.h"

Defines:

Symbol	Value
BCS_DEBUG	FALSE
MAX_BCS_SETTING	3000.0
MIN_BCS_SETTING	0.0
BCS_BOOTUP_TIME	1.2 seconds
BCS_BOOTUP_TICKS	((int)(BCS_BOOTUP_TIME*TICKS_PER_SECOND))
ISU_DELTA_ELEV	(deg_to_rad(0.2))

int declarations and initialization:

bcs_bootup_timer = NULL_TIMER	
bcs_booted_up = FALSE	either TRUE or FALSE
ammo_type_indexed	can be munition_US_M791, munition_USM792, or empty
bcs_null_status = FALSE	

REAL declarations:

apds_yb[10]	
apds_zb[10]	
heat_yb[10]	
heat_zb[10]	
bcs_range	in meters
super_elevation	sin(super elevation)
flight_time	seconds

char declarations:
bcs_range_str[5]

Pointer to char declarations and initialization:
bcs_range_format = "%04d"

Procedure declarations:
bcs_set_ballistics_computer()
turret_set_super_elevation()

2.3.3.1.1 bcs_init

This routine does a one-time initialization of the ballistics computer system.

Calls	
Function	**Where Described**
ballistics_load_parameter_file	Section 2.5.2.3.3
bcs_set_ballistics_computer	Section 2.3.3.1.7
bcs_ammo_index_no_round	Section 2.3.3.1.5
turret_set_super_elevation	Section 2.3.2.3.6.1.1.31

Table 2.3-82: bcs_init Information.

2.3.3.1.2 bcs_simul

This routine performs the ballistics computer system simulation on a tick by tick basis.

Calls	
Function	**Where Described**
timers_get_in_use_status	Section 2.6.3.7.1
timers_get_ticking_status	Section 2.6.3.20.1
timers_free_timer	Section 2.6.3.5.1
timers_set_null_timer	Section 2.6.3.14.1
controls_turret_power_system_on	Section 2.3.2
turret_set_super_elevation	Section 2.3.6.1.1.31

Table 2.3-83: bcs_simul Information.

2.3.3.1.3 bcs_ammo_index_hei_25

This routine notifies the ballistics computer system that the 25mm gun is selected with hei ammunition.

Calls	
Function	**Where Described**
bcs_set_ballistics_computer	Section 2.3.3.1.7

Table 2.3-84: bcs_ammo_index_hei_25 Information.

2.3.3.1.4 bcs_ammo_index_apds_25

This routine notifies the ballistics computer system that the 25mm gun is selected with apds ammunition.

Calls	
Function	**Where Described**
bcs_set_ballistic_computer	Section 2.3.3.1.7

Table 2.3-85: bcs_ammo_index_apds_25 Information.

2.3.3.1.5 bcs_ammo_index_no_round

This routine notifies the ballistics computer system that the 25mm gun is no longer selected. This routine is called when the TOW is selected, or when no ammo is selected. The ballistics computer is not set from this routine; the range string displayed in the ISU is cleared.

2.3.3.1.6 bcs_range_is

This routine is called by the controls module to notify the ballistics computer system of a new range value. The bcs performs error checking on the value.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
range_from_switch	REAL	sim_types.h
Errors		
Error Name	**Reason for Error**	
PANIC--invalid range ... in bcs	The variable range_from_switch has an unexpected value.	
Calls		
Function	**Where Described**	
bcs_set_ballistic_computer	Section 2.3.3.1.7	

Table 2.3-86: bcs_range_is Information.

2.3.3.1.7 bcs_set_ballistics_computer

This routine performs simulation of the on-board ballistics computer. It calculates the flight time and super elevation of the indexed ammunition type, given the bcs_range.

Errors	
Error Name	**Reason for Error**
BCS ERROR: Unknown ammo type	The variable ammo_type_indexed has an unexpected value.
Calls	
Function	**Where Described**
ballistics_calc_time	Section 2.5.2.1.1
ballistics_calc_se	Section 2.5.2.1.2
turret_set_super_elevation	Section 2.3.6.1.1.31

Table 2.3-87: bcs_set_ballistics_computer Information.

2.3.3.1.8 bcs_get_super_elevation

This routine returns the super elevation.

Return Values		
Return Value	**Type**	**Meaning**
super_elevation	REAL	The super elevation

Table 2.3-88: bcs_get_super_elevation Information.

2.3.3.1.9 bcs_get_range

This routine returns the bcs range.

Return Values		
Return Value	**Type**	**Meaning**
bcs_range	REAL	The bcs range

Table 2.3-89: bcs_get_range Information.

2.3.3.1.10 bcs_get_time_of_flight

This routine returns the flight time of the projectile.

Return Values		
Return Value	**Type**	**Meaning**
flight_time	REAL	The flight time of the projectile

Table 2.3-90: bcs_get_time_of_flight Information.

2.3.3.1.11 bcs_get_ammo_type_indexed

This routine returns the type of ammunition selected.

Return Values		
Return Value	**Type**	**Meaning**
ammo_type_selected	int	The selected ammunition type

Table 2.3-91: bcs_get_ammo_type_indexed Information.

2.3.3.1.12 bcs_get_range_str

This routine is called by the graphics interface to get the range to be displayed on the screen of the gunner's view (100s of meters).

Return Values		
Return Value	**Type**	**Meaning**
bcs_range_str	pointer to char	The bcs range string

Table 2.3-82: bcs_get_range_str Information.

2.3.3.1.13 bcs_turn_computer_on

This routine is called by controls to boot the ballistics computer system when turret power is turned on.

Calls	
Function	**Where Described**
timers_get_timer	Section 2.6.3.6.1

Table 2.3-93: bcs_turn_computer_on Information.

2.3.3.1.14 bcs_turn_computer_off

This routine turns off the ballistics computer system. This routine is called when turret power is turned off.

Calls	
Function	**Where Described**
controls_turret_power_system_off	Section 2.3.2
turret_set_super_elevation	Section 2.3.6.1.1.31
timers_set_null_timer	Section 2.6.3.14.1
timers_free_timer	Section 2.6.3.5.1

Table 2.3-94: bcs_turn_computer_off Information.

2.3.3.1.15 bcs_str_null

If bcs_null_status is TRUE, this routine sets it FALSE and returns TRUE. If bcs_null_status is FALSE, this routine returns FALSE.

Return Values		
Return Value	Type	Meaning
TRUE	int	bcs_null_status was TRUE, is now FALSE.
FALSE	int	bcs_null_status was FALSE, is not changed.

Table 2.3-95: bcs_str_null Information.

2.3.3.2 m2_weapons.c
(./simnet/release/src/vehicle/m2/src/m2_weapons.c [m2_weapons.c])

Miscellaneous weapons functions for the M2 are performed by this CSU.

Includes:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"basic.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"failure.h"
"libfail.h"
"librva.h"
"libevent.h"
"libmatrix.h"
"libmath.h"
"libturret.h"
"libhull.h"
"libimps.h"
"libkin.h"
"libsound.h"
"librepair.h"
"libnetworkh"
"libball.h"
"libmissile.h"
"libmap.h"
"miss_tow.h"
"mun_type.h"
"m2_mem_dfn.h"
"m2_ammo.h"
"m2_ammo_df.h"
"m2_bcs.h"

"m2_dtrain.h"
"m2_elecsys.h"
"m2_firectl.h"
"m2_turret.h"
"m2_sound_dfn.h"
"m2_sound.h"
"m2_weapons.h"

Defines:

Symbol	Value	Description
WEAPONS_EMPTY	-1	
RAD_STD_DEV	(mil_to_rad(0.3))	0.3 mils
TOW_LAUNCH_X	(-0.9)	meters to side of turret center
TOW_LAUNCH_Y	0.5	meters from center of turret
TOW_LAUNCH_Z	1.2	meters above gun axes
TOW_TIMER_DELAY	22	1.5 second tow delay
GUN_BREECH_X	0.15228	meters from base of hull
GUN_BREECH_Y	(-0.53299)	meters from base of hull
GUN_BREECH_Z	2.275	meters from base of hull
GUN_LENGTH	2.83299	meters from turret center
GUN_SIGHT_X	0.0	meters from turret center
GUN_SIGHT_Y	0.96447	meters from turret center
GUN_SIGHT_Z	0.0	meters from turret center
EFFECT_OFFSET	3.0	
LOW_FIRE_RATE_CHANGE_MASK	9	100rounds per minute
HIGH_FIRE_RATE_CHANGE_MASK	5	180rounds per minute
MAX_MISSILES	2	missiles in flight

TOW_MISSILE declarations:
missiles[MAX_MISSILES]

VECTOR declarations and initialization:

missile_fire_pos	location of launch point in turret coordinates
turret_center	location of turret center in hull coordinates
rel_sight_pos	location of sight in turret coordinates

int declarations and initialization:
ammo_type_loaded
ammo_type_in_flight
tow_timer = 0
tow_launcher_status = WORKING
high_fire_rate = FALSE
multi_shot_mode = FALSE
trigger_is_pulled = FALSE
shot_misfired = FALSE

ObjectType declarations:
ammo_type_selected

2.3.3.2.1 weapons_missile_is_launched

This routine launches the missiles (up to the maximum number of missiles).

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
launch_point	VECTOR	sim_types.h

Calls	
Function	**Where Described**
vec_mat_mul	Section 2.6.2.56.1
vec_add	Section 2.6.2.57.1
kinematics_get_h_to_w	Section 2.5.8.1.2
kinematics_get_o_to_h	Section 2.5.8.1.4
missile_tow_fire	Section 2.5.3.13.2
ammo_weapon_is_fired	Section 2.3.5.1.56
sound_make_const_sound	Section 2.1.3.1.2
suspension_gun_fired	Section 2.5.6.1.1
turret_to_hull_cos	
turret_to_hull_sin	

Table 2.3-96: weapons_missile_is_launched Information.

2.3.3.2.2 tow_fired_check

This routine is called each tick to see if launcher conditions are met. If so, then missiles are launched.

Calls	
Function	**Where Described**
firectl_tow_ready_to_fire	Section 2.3.2.2.3.12
map_get_network_type_from_ammo_entry	Section 2.6.11.2.3
electsys_tow_request	Section 2.3.6.3.1.21
weapons_missile_is_launched	Section 2.3.3.2.1

Table 2.3-97: tow_fired_check Information.

2.3.3.2.3 weapons_fire_round

This routine fires a round of the specified *ammo*. First, fire position in world coordinates is computed. The gun velocity vector is computed. Random biases are computed and a biased gun-to-world matrix is recorded. The gun is fired, notifying both suspension and ballistics. The routine calculates the gun muzzle position, painting the muzzle blast. The network is notified of the weapon firing.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
gun_position	VECTOR	sim_types.h
gun_velocity	VECTOR	sim_types.h
bias_vector	VECTOR	sim_types.h
gun_to_world	T_MATRIX	sim_types.h
f_pos[3]	float	Standard
f_gun_velocity[3]	float	Standard
round_id	int	Standard
gun_muzzle	VECTOR	sim_types.h
f_gun_muzzle[3]	float	Standard
Calls		
Function	**Where Described**	
vec_mat_mul	Section 2.6.2.56.1	
kinematics_get_h_to_w	Section 2.5.8.1.2	
vec_add	Section 2.6.2.57.1	
kinematics_get_o_to_h	Section 2.5.8.1.4	
vec_scale	Section 2.6.2.64.1	
kinematics_get_d_pos	Section 2.5.8.1.7	
d2f_vec_copy	Section 2.6.2.2.1	
bivariant_normal_distribution	Section 2.6.1.1.1	
turret_get_gun_to_world	Section 2.3.6.1.1.28	
event_get_eventid	Section 2.6.9.1.2	
suspension_gun_fired	Section 2.5.6.1.1	
turret_to_hull_cos		
turret_to_hull_sin		
ballistics_fire_a_round	Section 2.5.2.2.1	
map_get_tracer_from_ammo_entry	Section 2.6.11.2.9	
vec_init	Section 2.6.2.61.1	
network_send_shell_fire_pkt	Section 2.1.1.3.1.57.1	
turret_get_turret_slew_rate	Section 2.3.6.1.1.4	
bcs_get_ammo_type_indexed	Section 2.2.3.1.18	

Table 2.3-98: weapons_fire_round Information.

2.3.3.2.4 weapons_fire

This routine fires the weapon. It first gets the ammo type, then determines the rate of fire change mask. It checks for the trigger being pulled in single-shot or multi-shot mode. It fires the weapon for the appropriate ammo. It makes the sound of firing, and keeps track of failures and repairs.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
fire_rate_change_mask	int	Standard
gunnery_frame_counter	int	Standard

Calls	
Function	**Where Described**
ammo_round_selected_status	Section 2.3.5.1.26
map_get_ammo_entry_from_network_type	Section 2.6.11.2.1
ammo_round_loaded_status	Section 2.3.5.1.27
firectl_25mm_ready_to_fire	Section 2.3.2.2.3.11
ammo_bolt_position_status	Section 2.3.5.1.53
ammo_low_ammo_ready_to_fire	Section 2.3.5.1.51
electsys_25mm_gun_request	Section 2.3.6.3.1.24
fail_break_system	Section 2.5.4.8.1
sound_make_const_sound	Section 2.1.3.1.2
sfail_event_occurred	Section 2.5.4.21.1
ammo_weapon_is_fired	Section 2.3.5.1.56
sound_of_main_gun_firing	Section 2.1.3.3.9
weapons_fire_round	Section 2.3.3.2.3
firectl_tow_ready_to_fire	Section 2.3.2.2.3.12
firectl_weapon_ready	Section 2.3.2.2.3.10
ammo_misfire_lock_status	Section 2.3.5.1.54
repair_fix_failure	Section 2.5.4.19.5

Table 2.3-99: weapons_fire Information.

2.3.3.2.5 weapons_simul

This routine performs the tick-by-tick simulation of the weapons.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
found_missile	int	Standard
vehicle_moving	int	Standard
sight_pos	VECTOR	sim_types.h
Calls		
Function	**Where Described**	
tow_fired_check	Section 2.3.3.2.2	
vec_mat_mul	Section 2.6.2.56.1	
vec_add	Section 2.6.2.57.1	
kinematics_get_h_to_w	Section 2.5.8.1.2	
kinematics_get_o_to_h	Section 2.5.8.1.4	
drivetrain_get_vehicle_speed	Section 2.3.6.2.4.8	
missile_tow_cut_wire	Section 2.5.3.13.5	
missile_tow_fly	Section 2.5.3.13.3	

Table 2.3-100: weapons_simul Information.

2.3.3.2.6 weapons_init

This routine initializes the weapons system.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
time_temp	long	Standard
Calls		
Function	**Where Described**	
missile_tow_init	Section 2.5.3.13.1	
missile_util_init	Section 2.5.3.27.1	

Table 2.3-101: weapons_init Information.

2.3.3.2.7 weapons_set_low_fire_rate

This routine is called from the CIG and Handles modules. It sets the multi shot mode flag to TRUE, signifying that the weapon is in multi-shot mode.

2.3.3.2.8 weapons_trigger_is_released

This routine releases the trigger.

2.3.3.2.9 weapons_cut_any_tow_wires

This routine cuts any tow wires.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
Calls		
Function	**Where Described**	
missile_tow_cut_wire	Section 2.5.3.13.5	

Table 2.3-102: weapons_cut_any_tow_wires Information.

2.3.3.2.10 weapons_set_high_fire_rate

This routine sets *high_fire_rate* and *multi_shot_mode* equal to TRUE, signifying that the weapon is in high fire multi-shot mode.

2.3.3.2.11 weapons_set_single_shot_mode

This routine sets *multi_shot_mode* equal to FALSE, signifying that the weapon is in signle shot mode.

2.3.3.2.12 weapons_trigger_is_pulled

This routine sets *trigger_is_pulled* equal to TRUE, signifying that the trigger is pulled.

2.3.3.2.13 weapons_trigger_status

This routine returns the trigger status.

Return Values		
Return Value	**Type**	**Meaning**
trigger_is_pulled	int	If equal to TRUE, the trigger is pulled; if equal to FALSE, the trigger is released.

Table 2.3-103: weapons_trigger_status Information.

2.3.3.2.14 weapons_shot_misfired

This routine causes a shot misfire failure.

2.3.3.2.15 weapons_break_tow_launcher

This routine causes the tow launcher to fail.

2.3.3.2.16 weapons_repair_tow_launcher

This routine repairs the tow launcher.

2.3.3.2.17 weapons_misfire_corrected

This routine repairs a weapons misfire.

Calls	
Function	**Where Described**
ammo_misfire_corrected	Section 2.3.5.1.58

Table 2.3-104: weapons_misfire_corrected Information.

2.3.3.2.18 weapons_vehicle_rolled

This routine is not used in the version 6.6 release.

2.3.3.2.19 weapons_vehicle_unrolled

This routine is not used in the version 6.6 release.

2.3.3.2.20 weapons_download_ballistics_tables

This routine downloads the ballistics trajectory data file for the ammo types used in the M2 tank.

Calls	
Function	**Where Described**
ballistics_load_trajectory_file	Section 2.5.2.3.1

Table 2.3-105: weapons_download_ballistics_tables Information.

2.3.3.2.21 weapons_keybrd_fire

This routine fires a round, responding to a command from the keyboard.

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

Calls	
Function	**Where Described**
weapons_fire_round	Section 2.3.3.2.3

Table 2.3-106: weapons_keybrd_fire Information.

2.3.4 M2 Failures

Failure generation in SIMNET allows the introduction of simulated physical failures and degradations of a vehicle's capabilities. This involves the generation of a variety of different kinds of failures and the presentation of these failures to either the crew of a manned simulator, or the operator of a computer controlled vehicle. Failures are divided into categories that indicate the method used for failure generation. The three categories are combat damage, stochastic failure, and deterministic failure.

Combat Damage

During combat, a vehicle receives combat information messages from the network. This information comes in two different forms. First, an 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.

Stochastic Failures

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.

Deterministic Failures

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.

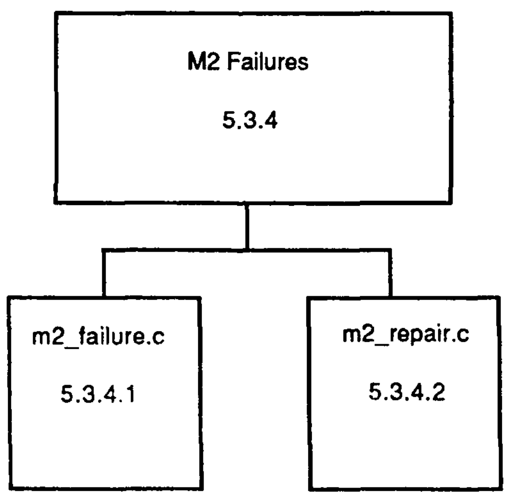

Figure 2.3-5: Structure of the M2 Failures CSC.

The CSU's listed below provide the routines for failure simulation.

m2_failure.c
m2_repair.c

2.3.4.1 m2_failure.c

(./simnet/release/src/vehicle/m2/src/m2_failure.c [m2_failure.c])

The vehicle specific failures functions are found in this CSU.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"pro_sim.h"
"mun_type.h"
"pro_data.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"libfail.h"
"failure.h"
"status.h"
"status_m2.h"
"repair_m2.h"
"libnetwork.h"
"librepair.h"
"libhull.h"
"bigwheel.h"
"libsound.h"
"libmain.h"
"librva.h"
"m2_cntrl.h"
"m2_dtrain.h"
"m2_elecsys.h"
"m2_engine.h"
"m2_engfail.h"
"m2_failure.h"
"m2_fuelsys.h"
"m2_handles.h"
"m2_sound_dfn.h"
"m2_trans.h"
"m2_turret.h"
"m2_vision.h"
"m2_weapons.h"
"m2_rep_map.h"

Defines:

NUM_SUB_SYSTEMS
MOBILITY_SYSTEM
FIREPOWER_SYSTEM
SUPPORT_SYSTEM
ACTUAL_NUM_LEVELS
USED_NUM_LEVELS
MAINT_L_1_MMBF
MAINT_L_2_MMBF
MAINT_L_3_MMBF
MAINT_L_4_MMBF
MAINT_L_5_MMBF

Declarations:

MAINT_LEVEL_ARRAY

2.3.4.1.1 fail_init

This routine initializes the failures system. The combat failures and combat failure table are initialized, and the stochastic failures and stochastic damages table are initialized. The repairs are initialized and the collision damages are initialized.

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

Calls	
Function	**Where Described**
failure_collision_damages	Section 2.3.4.1.2
cfail_init	Section 2.5.4.5.1
fail_table_init	Section 2.5.4.11.1
sfail_init	Section 2.5.4.23.1
get_sdamage_file	Section 2.5.1.2.7
lrepair_init	Section 2.5.4.19.1
collision_init	Section 2.5.10.5.1

Table 2.3-107: fail_init Information.

2.3.4.1.2 failure_collision_damages

This routine is called by the collision code when a collision is detected. Parameters are represented as follows:

direction -- The direction the collision came from: RIGHT_WHEEL, LEFT_WHEEL, or REAR_WHEEL, as defined in /simnet/vehicle/libbigwh/bigwh_loc.h

cause -- The cause of the collision. *cause* can be either a hash id corresponding to the vehicle collided with or -2 (signifying a ground collision)

event_id -- The event id corresponding to the collision

Parameters		
Parameter	**Type**	**Where Typedef Declared**
direction	int	Standard
cause	int	Standard
event_id	long int	Standard
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
vid	pointer to VehicleID	basic.h
Return Values		
Return Value	**Type**	**Meaning**
0	int	Successful

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2
rva_get_veh_id	Section 2.5.12.10.1
turret_collision_detected	Section 2.3.6.1.1.25
network_send_outta_my_way_mf	Section 2.1.1.3.1.8.1

Table 2.3-108: failure_collision_damages Information.

2.3.4.1.3 failure_check_cat_kill

This routine checks to see if a direct impact packet came from a mine. If the packet is from a mine, then the tank is destroyed. If the packet is not from a mine, then damages are calculated by calling **cfail_dir_fire_damages**() in libfail. Parameters are represented as follows:

ammo_type -- the ammo indix from libmap
hit_msg -- pointer to the Impact Variant PDU

Parameters		
Parameter	**Type**	**Where Typedef Declared**
hit_msg	pointer to ImpactVariant	p_sim.h
ammo_type	int	Standard

Calls	
Function	**Where Described**
fail_vehicle_is_destroyed	Section 2.5.4.9.2
cfail_dir_fire_damages	Section 02.5.4.3.1

Table 2.3-109: failure_check_cat_kill Information.

2.3.4.1.4 failure_check_indir_fire_damages

This routine checks to see if an indirect impact packet came from a mine. If the packet is from a mine and the detonation occurs within 10 meters of the tank, then the tank is destroyed. If the packet is not from a mine, then damages are calculated by calling **cfail_indirect_fire_damages()** in libfail. Parameters are represented as follows:

ammo_type -- the ammo indix from libmap
indir_fire_msg -- pointer to the Indirect Fire Variant PDU
r_squared -- The range from the detonation to the vehicle
detonation_num -- The detonation number within the Indirect Fire Variant PDU

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo_type	int	Standard
indir_fire_msg	pointer to IndirectFireVariant	p_sim.h
r_squared	REAL	sim_types.h
detonation_num	int	Standard

Calls	
Function	**Where Described**
fail_vehicle_is_destroyed	Section 2.5.4.9.2
cfail_indirect_fire_damages	Section 2.5.4.4.1
kinematics_get_h_to_o	Section 2.5.8.1.3
kinematics_get_w_to_h	Section 2.5.8.1.1

Table 2.3-110: failure_check_indir_fire_damages Information.

2.3.4.2 m2_repair.c

(./simnet/release/src/vehicle/m2/src/m2_repair.c [m2_repair.c])

The M2 specific repairs functions are found in this CSU.

Includes:

"stdio.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"pro_assoc.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"pro_data.h"
"pro_sim.h"
"repair_m2.h"
"timers_dfn.h"
"timers.h"
"libnetwork.h"
"m2_cntrl.h"
"m2_resupp.h"

Defines:

QUIET
REQUEST
MAX_REPAIR_ENTITIES

Declarations:

repair_vehicle
repair_vehicles_near_hear
num_repair_vehicles
repair_state
repair_timer_id
clear_repair_vehicles()
repair_quiet_state()
repair_request_state()
send_feed_me_packets_repair_vehicles()

2.3.4.2.1 repair_request

This routine processes repair requests that come in over the network from the Admin/Log console of the MCC. The MCC host times these repairs and informs the simulator that the repair is complete via a network message.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
event	int	Standard
agent	pointer to VehicleID	basic.h
code	int	Standard
originator	pointer to SimulationAddress	address.h
tid	TransactionIdentifier	p_assoc.h

Errors	
Error	**Reason for Error**
stderr, REPAIR: repair_request()	unknown repair state

Calls	
Function	**Where Described**
timers_free_timer	Section 2.6.3.5.1
repair_system_is_fixed	Section 2.5.4.19.4
send_repaired_pkt	Section 2.1.1.3.1.38.1

Table 2.3-111: repair_request Information.

2.3.4.2.2 repair_simul

This routine runs the repair simulation and is responsible for all self-repairs to the simulation The routine checks the vehicle's repair state and calls the appropriate routine for that state.

Errors	
Error	**Reason for Error**
stderr, REPAIR: repair_simul()	unknown repair state

Calls	
Function	**Where Described**
repair_quiet_state	Section 2.3.4.2.7
repair_request_state	Section 2.3.4.2.8
clear_repair_vehicles	Section 2.3.4.2.4

Table 2.3-112: repair_simul Information.

2.3.4.2.3 repair_init

This routine initializes the repair simulation. All repair vehicles are cleared, the repair state is set to QUIET, and the repair timer is set to NULL_TIMER.

Calls	
Function	**Where Described**
clear_repair_vehicles	Section 2.3.4.2.4

Table 2.3-113: repair_init Information.

2.3.4.2.4 clear_repair_vehicles

This routine clears the *repair_vehicles_near_here* flag to FALSE and the number of repair vehicles to zero.

2.3.4.2.5 repair_near_repair

This routine maintains the list of close vehicles which are repair vehicles. If any repair vehicles are on this list, the *repair_vehicles_near_here* flag is set to TRUE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v	pointer to VehicleID	basic.h

Table 2.3-114: repair_near_repair Information.

2.3.4.2.6 send_feed_me_packets_repair_vehicles

This routine sends a repair request packet to each of the repair vehicles on the network.

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

Calls	
Function	**Where Described**
network_send_feed_me_packet	Section 2.1.1.3.1.48.1

Table 2.3-115: send_feed_me_packets_repair_vehicles Information.

2.3.4.2.7 repair_quiet_state

This routine determines the vehicle's repair Finite State Machine's QUIET state. If the following conditions are BOTH TRUE:

- The resupply gating conditions are TRUE (the vehicle is alive, the vehicle is not moving, and no controls failures exist).
- There are repair vehicles nearby.

Then, send a feed me packet to the repair vehicles on the network, start the repair timer, and enter the REQUEST state. If any of the conditions are not met, remain in the QUIET state.

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	The vehicle is in the repair FSM QUIET state
REQUEST	int	The vehicle is in the repair FSM REQUEST state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.3.5.3.15
timers_get_timer	Section 2.6.3.6.1
send_feed_me_packets_repair_vehicles	Section 2.3.4.2.6

Table 2.3-116: repair_quiet_state Information.

2.3.4.2.8 repair_request_state

This routine determines the vehicle's repair Finite State Machine's REQUEST state. If EITHER of the following conditions are TRUE:

- The resupply gating conditions are FALSE (the vehicle is dead, the vehicle is moving, or a controls failure exists).
- There are no fuel carriers nearby.

Then, free the repair timer and enter the QUIET state. If none of the conditions are met and the resupply timer has expired, send another service request, restart the timer, and remain in the REQUEST state. If the resupply timer has not expired, remain in the REQUEST state.

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	The vehicle is in the repair FSM QUIET state
REQUEST	int	The vehicle is in the repair FSM REQUEST state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.3.5.3.15
timers_get_timer	Section 2.6.3.6.1
send_feed_me_packets_repair_vehicles	Section 2.3.4.2.6

Table 2.3-117: repair_request_state Information.

2.3.4.2.9 print_repair_status

This routine prints the repair status information.

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

Table 2.3-118: print_repair_status Information.

2.3.5 M2 Munitions Management

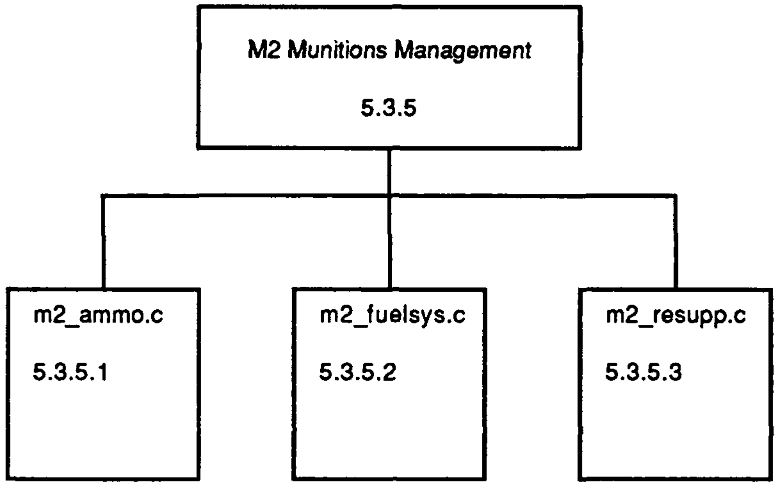

Figure 2.3-6: Structure of the M2 Munitions Management CSC.

The simulation monitors fuel and ammunition in the above files. The fuel system involves more than one fuel tank, and the fuel may transfer between these tanks within the vehicle. The simulation must also determine the states of all munitions on board a vehicle. This includes the availability, which ammunitions are loaded, and which ammunition is being fired. Resupply of all munitions (fuel and ammunition) is coordinated in the file "m2_resupp.c" for the associated vehicles. These files implement the functions necessary to resupply munitions from MCC vehicles (hemmits, etc.) and other vehicle simulations (m1 to m1, m2 to m2).

The following CSU's provide this functionality.

m2_ammo.c
m2_fuelsys.c
m2_resupp.c

2.3.5.1 m2_ammo.c

(./simnet/release/src/vehicle/m2/src/m2_ammo.c [m2_ammo.c])

This CSU monitors the M2's ammunition supply.

Includes:

"stdio.h"
"sim_dfns.h"
"sim_macros.h"
"sim_types.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"timers_dfn.h"
"timers.h"
"libnetwork.h"
"libsound.h"
"failure.h"
"libfail.h"
"m2_alpha.h"
"m2_alpha_df.h"
"m2_ammo.h"
"m2_ammo_df.h"
"m2_turr_mx.h"
"m2_turr_pn.h"
"m2_cntrl.h"
"m2_ctl_df.h"
"m2_launcher.h"
"m2_resupp.h"
"m2_sound_dfn.h"
"m2_tmrs_df.h"
"m2_weapons.h"
"m2_isu.h"
"m2_turret.h"
"m2_bcs.h"
"m2_firectl.h"
"mun_type.h"

Static Variable and Procedure Declarations:

Note that the hei and apds ammunition cans are stored as a sequence of bits: AP == 0, HE == 1, where each bit represents a box of 30 rounds. The Least Significant Bit represents the box closest to the barrel, and the Most Significant Bit represents the box farthest away from the barrel.

apds_can_quantity
apds_can_ammo_boxes
hei_can_quantity
hei_can_ammo_boxes
apds_stowed_quantity
hei_stowed_quantity
tow_stowed_quanity
dragon_stowed_quantity

missile1_val
missile2_val
m3_configuration_val

n_25_stowed
n_missile_stowed
n_dragon_stowed

displayed_stowed_quantity

hei_can_switch_val
apds_can_switch_val
ammo_reversed_val
tow_launcher_val
gps_mag_val
bolt_position
ammo_round_selected
ammo_round_loaded
ammo_round_indexed
ammo_round_chambered
ammo_round_last_selected
ammo_misfire_lock
low_ammo_val
low_ammo_lock_val
tow_test_in_progress
tow_test_val -- Where 0.0 == BEGIN and 1.0 == END
missile1_selected
missile2_selected
ammo_transfer_status -- For AMMO_RECIEVE_VAL, an ammunition transfer receive is valid; for AMMO_SEND_VAL, the ammunition transfer send is valid; for AMMO_NO_TRANSFER_VAL, no ammunition transfer is valid; for AMMO_INTERNAL_VAL, an internal ammunition transfer is valid; for AMMO
ammo_mgmt_round
resupply_counter
internal_resupply_in_progress

ammo_reversed_check()
ammo_indexed_check()
ammo_tow_test_check()
ammo_low_ammo_check()
ammo_tow_test_start()
ammo_two_test_stop()
ammo_get_missile_loaded()
ammo_get_apds_can_first_round()
ammo_get_hei_can_first_round()
ammo_remove_apds_can_round()
ammo_remove_hei_can_round()
ammo_get_apds_can_box()
ammo_get_hei_can_box()
ammo_hei_can_enough_room()

ammo_apds_can_enough_room()
ammo_25mm_stowage_enough_room()
ammo_two_tubes_enough_room()
ammo_tow_stowage_enough_room()
ammo_dragon_stowage_enough_room()
ammo_resupply_timeout_check()
ammo_ready_to_internal_resupply()
ammo_start_internal_resupply()
ammo_internal_resupply_start_check()
ammo_internal_resupply_abort_check()
ammo_rounds_on_board_check()
ammo_hei_stowage_enough_supply()
ammo_apds_stowage_enough_supply()
ammo_tow_stowage_enough_supply()
ammo_supply_empty_stowage()

2.3.5.1.1 ammo_init

This routine initalizes the ammo system.

Calls	
Function	**Where Described**
controls_bolt_position_sear	Section 2.3.2
fail_init_failure	Section 2.5.4.11.2

Table 2.3-119: ammo_init Information.

2.3.5.1.2 ammo_simul

This routine simulates the various functions of the ammunition system on a tick by tick basis. The routine checks the status of any internal or external ammunition transfers or resupplies. The routine monitors the types and quantities of ammo in location.

Calls	
Function	**Where Described**
ammo_reversed_check	Section 2.3.5.1.29
ammo_indexed_check	Section 2.3.5.1.31
ammo_tow_test_check	Section 2.3.5.1.46
ammo_low_ammo_check	Section 2.3.5.1.49
ammo_internal_resupply_start_check	Section 2.3.5.1.87
ammo_internal_resupply_abort_check	Section 2.3.5.1.88
ammo_resupply_timeout_check	Section 2.3.5.1.89
ammo_rounds_on_board_check	Section 2.3.5.1.90

Table 2.3-120: ammo_simul Information.

2.3.5.1.3 ammo_init_ammo_supply

This routine initializes the ammo system's ammunition supply. The routine determines whether the tank is configured as an M2 or an M3. The values which are passed for ammo supply quanitites are compared to the maximum numbers allowed for the tank configuration to make sure that the tank is not supplied with more ammo than it is allowed.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
temp_apds_can_quantity	int	Standard
temp_apds_can_ammo_boxes	int	Standard
temp_hei_can_quantity	int	Standard
temp_apds_can_ammo_boxes	int	Standard
temp_apds_stowed_quantity	int	Standard
temp_hei_stowed_quantity	int	Standard
temp_tow_stowed_quantity	int	Standard
temp_dragon_stowed_quantity	int	Standard
temp_missile1_val	int	Standard
temp_missile2_val	int	Standard
temp_m3_configuration_val	int	Standard

Table 2.3-121: ammo_init_ammo_supply Information.

2.3.5.1.4 ammo_get_apds_can_quantity

This routine returns the quantity of armor piercing discarding sabot (apds) canisters in the tank.

Return Values		
Return Value	**Type**	**Meaning**
apds_can_quantity	int	Quantity of apds cans in the tank.

Table 2.3-122: ammo_get_apds_can_quantity Information.

2.3.5.1.5 ammo_get_apds_can_ammo_boxes

This routine returns the quantity of apds canisters in the two ammunition boxes.

Return Values		
Return Value	**Type**	**Meaning**
apds_can_ammo_boxes	int	Quantity of apds cans in the ammo boxes.

Table 2.3-123: ammo_get_apds_can_ammo_boxes Information.

2.3.5.1.6 ammo_get_hei_can_quantity

This routine returns the quantity of high explosive anti tank (hei) canisters in the tank.

Return Values		
Return Value	Type	Meaning
hei_can_quantity	int	Quantity of hei cans in the tank.

Table 2.3-124: ammo_get_hei_can_quantity Information.

2.3.5.1.7 ammo_get_hei_can_ammo_boxes

This routine returns the quantity of hei canisters in the two ammunition boxes.

Return Values		
Return Value	Type	Meaning
hei_can_ammo_boxes	int	Quantity of hei cans in the ammo boxes.

Table 2.3-125: ammo_get_hei_can_ammo_boxes Information.

2.3.5.1.8 ammo_get_apds_stowed_quantity

This routine returns the quantity of apds in stowage.

Return Values		
Return Value	Type	Meaning
apds_stowed_quantity	int	Quantity of apds in stowage.

Table 2.3-126: ammo_get_apds_stowed_quantity Information.

2.3.5.1.9 ammo_get_hei_stowed_quantity

This routine returns the quantity of hei in stowage.

Return Values		
Return Value	Type	Meaning
hei_stowed_quantity	int	Quantity of hei in stowage.

Table 2.3-127: ammo_get_hei_stowed_quantity Information.

2.3.5.1.10 ammo_get_tow_stowed_quantity

This routine returns the number of TOW missiles in stowage.

Return Values		
Return Value	Type	Meaning
tow_stowed_quantity	int	Quantity of TOW missiles in stowage.

Table 2.3-128: ammo_get_tow_stowed_quantity Information.

2.3.5.1.11 ammo_get_dragon_stowed_quantity

This routine returns the number of Dragon missiles in stowage.

Return Values		
Return Value	**Type**	**Meaning**
dragon_stowed_quantity	int	Quantity of Dragon missiles in stowage.

Table 2.3-129: ammo_get_dragon_stowed_quantity Information.

2.3.5.1.12 ammo_get_missile1_val

This routine returns whether the missile 1 launcher is being used.

Return Values		
Return Value	**Type**	**Meaning**
missile1_val	int	If 0, missile 1 tube is invalid; If 1, missile 1 tube is valid.

Table 2.3-130: ammo_get_missile1_val Information.

2.3.5.1.13 ammo_get_missile2_val

This routine returns whether the missile 2 launcher is being used.

Return Values		
Return Value	**Type**	**Meaning**
missile2_val	int	If 0, missile 2 tube is invalid; If 1, missile 2 tube is valid.

Table 2.3-131: ammo_get_missile2_val Information.

2.3.5.1.14 ammo_get_m3_configuration_val

This routine returns whether the vehicle is configured as an M2 or an M3 vehicle.

Return Values		
Return Value	**Type**	**Meaning**
m3_configuration_val	int	If 0, vehicle is an M2; If 1, vehicle is an M3.

Table 2.3-132: ammo_get_apds_can_quantity Information.

2.3.5.1.15 ammo_hei_can_hei_on

When the hei ammo box switch is set to hei, this routine sets the value of *hei_can_switch_val* equal to *munition_US_M792*. This signifies that the hei ammo box contains hei rounds.

2.3.5.1.16 ammo_hei_can_hei_off

When the hei ammo box switch is set to apds, this routine sets the value of *hei_can_switch_val* equal to *munition_US_M791*. This signifies that the hei ammo box contains apds rounds.

2.3.5.1.17 ammo_apds_can_hei_on

When the apds ammo box switch is set to hei, this routine sets the value of *apds_can_switch_val* equal to *munition_US_M792*. This signifies that the apds ammo box contains hei rounds.

2.3.5.1.18 ammo_apds_can_hei_off

When the apds ammo box switch is set to apds, this routine sets the value of *apds_can_switch_val* equal to *munition_US_M791*. This signifies that the apds ammo box contains apds rounds.

2.3.5.1.19 ammo_mgmt_receive_pushed

This routine sets the ammo transfer status when the Ammunition Management Receive button is pressed. Any type of resupply in progress is stopped. If a resupply receive was already in progress, it is discontinued and the transfer status is set to AMMO_NO_TRANSFER_VAL. Otherwise, the receive is enabled by setting transfer status to AMMO_RECEIVE_VAL.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.3.5.1.76

Table 2.3-133: ammo_mgmt_receive_pushed Information.

2.3.5.1.20 ammo_mgmt_send_pushed

This routine sets the ammo transfer status when the Ammunition Management Send button is pressed. Any type of resupply in progress is stopped. If a resupply send was already in progress, it is discontinued and the transfer status is set to AMMO_NO_TRANSFER_VAL. Otherwise, the send is enabled by setting the transfer status to AMMO_SEND_VAL.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.3.5.1.76

Table 2.3-134: ammo_mgmt_send_pushed Information.

2.3.5.1.21 ammo_mgmt_internal_pushed

This routine sets the ammo transfer status when the Ammunition Management Internal button is pressed. Any type of resupply in progress is stopped. If a internal ammo transfer was already in progress, it is stopped and the transfer status is set to AMMO_NO_TRANSFER_VAL. Otherwise, the internal transfer is enabled by setting the transfer status to AMMO_INTERNAL_VAL.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.3.5.1.76

Table 2.3-135: ammo_mgmt_internal_pushed Information.

2.3.5.1.22 ammo_mgmt_hei_pushed

This routine causes ammo from the hei box to be transferred when the Ammunition Management HE button is pressed and either the Send, Receive, or Internal pushbutton lamp is on.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.3.5.1.76

Table 2.3-136: ammo_mgmt_hei_pushed Information.

2.3.5.1.23 ammo_mgmt_apds_pushed

This routine causes ammo from the apds to be transferred when the Ammunition Management AP button is pressed and either the Send, Receive, or Internal pushbutton lamp is on.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.3.5.1.76

Table 2.3-137: ammo_mgmt_apds_pushed Information.

2.3.5.1.24 ammo_mgmt_tow_pushed

This routine causes TOW missiles to be transferred when the Ammunition Management TOW button is pressed and either the Send, Receive, or Internal pushbutton lamp is on.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.3.5.1.76

Table 2.3-138: ammo_mgmt_tow_pushed Information.

2.3.5.1.25 ammo_mgmt_dragon_pushed

This routine causes dragon missiles to be transferred when the Ammunition Management DRAGON button is pressed and either the Send, Receive, or Internal pushbutton lamp is on.

Calls	
Function	**Where Described**
ammo_stop_resupply	Section 2.3.5.1.76

Table 2.3-139: ammo_mgmt_dragon_pushed Information.

2.3.5.1.26 ammo_round_selected_status

This routine returns the ammo box which the user selected by pressing one of the Ammunition Management pushbuttons.

Return Values		
Return Value	**Type**	**Meaning**
(ObjectType) ammo_round_selected	ObjectType	The selected ammo round.

Table 2.3-140: ammo_round_selected_status Information.

2.3.5.1.27 ammo_round_loaded_status

This routine returns the ammunition which is loaded.

Return Values		
Return Value	**Type**	**Meaning**
(ObjectType) ammo_round_loaded	ObjectType	The loaded ammo round.

Table 2.3-141: ammo_round_loaded_status Information.

2.3.5.1.28 ammo_round_indexed_status

This routine returns the type of ammo round in the selected ammo box.

Return Values		
Return Value	**Type**	**Meaning**
(ObjectType) ammo_round_indexed	ObjectType	The ammo which is in the selected ammo box.

Table 2.3-142: ammo_round_indexed_status Information.

2.3.5.1.29 ammo_reversed_check

On a tick by tick basis, this routine is called to check whether the ammunition in the ammo cans is reversed. If the hei ammo box HE/AP switch is in the AP position (the box contains apds rounds), or if the apds ammo box HE/AP switch is in the HE position (the box contains hei rounds), the Ammo Reversal Indicator, located on the Gunner's Control Panel, will turn on.

Calls	
Function	**Where Described**
controls_gunner_ammo_reversed_on	Section 2.3.2
controls_gunner_ammo_reversed_off	Section 2.3.2

Table 2.3-143: ammo_reversed_check Information.

2.3.5.1.30 ammo_reversed_status

This routine returns *ammo_reversed_v al.* If this flag is ON, either the hei ammo box HE/AP switch is in the AP position or the apds ammo box HE/AP switch is in the HE position.

Return Values		
Return Value	**Type**	**Meaning**
ammo_reversed_val	int	If ON, the ammo is reversed; If OFF, the ammo is not reversed.

Table 2.3-144: ammo_reversed_status Information.

2.3.5.1.31 ammo_indexed_check

On a tick by tick basis, this routine is called to check which ammo type is contained in the selected ammo box. The actual type of ammo in the selected ammunition box is called the indexed ammo. This routine notifies the ballistics computer system (BCS) of the indexed ammo.

Calls	
Function	**Where Described**
bcs_ammo_index_hei_25	Section 2.3.3.1
bcs_ammo_index_apds_25	Section 2.3.3.1
bcs_ammo_index_no_round	Section 2.3.3.1

Table 2.3-145: ammo_indexed_check Information.

2.3.5.1.32 ammo_ap_ss_pushed

When the AP Single Shot button is pushed, this routine causes the system to be reset for apds ammo in single shot mode. The system cannot be reset if the weapon is in the process of being fired already (the trigger is pulled). If the weapon is not being fired, apds ammo is selected and loaded, and the weapon is set for single shot mode.

Calls	
Function	**Where Described**
weapons_trigger_status	Section 2.3.3.2.13
controls_round_select_ap_ss	Section 2.3.2
weapons_set_single_shot_mode	Section 2.3.3.2.11
isu_round_select_25mm	Section 2.3.6.3.3.5
turret_tow_movement_off	Section 2.3.6.1.1.29
ammo_tow_test_stop	Section 2.3.5.1.48
ammo_get_apds_can_first_round	Section 2.3.5.1.62

Table 2.3-146: ammo_ap_ss_pushed Information.

2.3.5.1.33 ammo_he_ss_pushed

When the HE Single Shot button is pushed, this routine causes the system to be reset for hei ammo in single shot mode. The system cannot be reset if the weapon is in the process of being fired already (the trigger is pulled). If the weapon is not being fired, hei ammo is selected and loaded, and the weapon is set for single shot mode.

Calls	
Function	**Where Described**
weapons_trigger_status	Section 2.3.3.2.13
controls_round_select_he_ss	Section 2.3.2
weapons_set_single_shot_mode	Section 2.3.3.2.11
isu_round_select_25mm	Section 2.3.6.3.3.5
turret_tow_movement_off	Section 2.3.6.1.1.29
ammo_tow_test_stop	Section 2.3.5.1.48
ammo_get_hei_can_first_round	Section 2.3.5.1.63

Table 2.3-147: ammo_he_ss_pushed Information.

2.3.5.1.34 ammo_ap_lo_pushed

When the AP Low Fire Rate button is pushed, this routine causes the system to be reset for apds ammo in low fire rate mode. The system cannot be reset if the weapon is in the process of being fired already (the trigger is pulled). If the weapon is not being fired, apds ammo is selected and loaded, and the weapon is set for the low fire rate.

Calls	
Function	**Where Described**
weapons_trigger_status	Section 2.3.3.2.13
controls_round_select_ap_'o	Section 2.3.2
weapons_set_low_fire_rate	Section 2.3.3.2.7
isu_round_select_25mm	Section 2.3.6.3.3.5
turret_tow_movement_off	Section 2.3.6.1.1.29
ammo_tow_test_stop	Section 2.3.2.5.1.48
ammo_get_apds_can_first_round	Section 2.3.5.1.62

Table 2.3-148: ammo_ap_lo_pushed Information.

2.3.5.1.35 ammo_he_lo_pushed

When the HE Low Fire Rate button is pushed, this routine causes the system to be reset for hei ammo in low fire rate mode. The system cannot be reset if the weapon is in the process of being fired already (the trigger is pulled). If the weapon is not being fired, hei ammo is selected and loaded, and the weapon is set for the low fire rate.

Calls	
Function	**Where Described**
weapons_trigger_status	Section 2.3.3.2.13
controls_round_select_he_lo	Section 2.3.2
weapons_set_low_fire_rate	Section 2.3.3.2.7
isu_round_select_25mm	Section 2.3.6.3.3.5
turret_tow_movement_off	Section 2.3.6.1.1.29
ammo_tow_test_stop	Section 2.3.5.1.48
ammo_get_hei_can_first_round	Section 2.3.5.1.63

Table 2.3-149: ammo_ap_ss_pushed Information.

2.3.5.1.36 ammo_ap_hi_pushed

When the AP High Fire Rate button is pushed, this routine causes the system to be reset for apds ammo in high fire rate mode. The system cannot be reset if the weapon is in the process of being fired already (the trigger is pulled). If the weapon is not being fired, apds ammo is selected and loaded, and the weapon is set for the high fire rate.

Calls	
Function	**Where Described**
weapons_trigger_status	Section 2.3.3.2.13
controls_round_select_ap_hi	Section 2.3.2
weapons_set_high_fire_rate	Section 2.3.3.2.10
isu_round_select_25mm	Section 2.3.6.3.3.5
turret_tow_movement_off	Section 2.3.6.1.1.29
ammo_tow_test_stop	Section 2.3.5.1.48
ammo_get_apds_can_first_round	Section 2.3.6.1.62

Table 2.3-150: ammo_ap_hi_pushed Information.

2.3.5.1.37 ammo_he_hi_pushed

When the HE High Fire Rate button is pushed, this routine causes the system to be reset for hei ammo in high fire rate mode. The system cannot be reset if the weapon is in the process of being fired already (the trigger is pulled). If the weapon is not being fired, hei ammo is selected and loaded, and the weapon is set for the high fire rate.

Calls	
Function	**Where Described**
weapons_trigger_status	Section 2.3.3.2.13
controls_round_select_he_hi	Section 2.3.2
weapons_set_high_fire_rate	Section 2.3.3.2.10
isu_round_select_25mm	Section 2.3.6.3.3.5
turret_tow_movement_off	Section 2.3.6.1.1.29
ammo_tow_test_stop	Section 2.3.5.1.48
ammo_get_hei_can_first_round	Section 2.3.5.1.63

Table 2.3-151: ammo_he_hi_pushed Information.

2.3.5.1.38 ammo_tow_select_pushed

When the TOW Select button is pushed, this routine causes the system to be reset for firing a TOW missile. In order for the TOW missile to be selected, the TOW launcher must be on, the gunners primary sight must be in 12X magnification, and the weapon must not be in the process of firing, and the TOW missile must not be already selected. If these conditions are met, the TOW missile is selected, the weapon is set for single shot mode, and the TOW test is started.

Calls	
Function	**Where Described**
weapons_trigger_status	Section 2.3.3.2.13
controls_round_select_tow	Section 2.3.2
weapons_set_single_shot_mode	Section 2.3.3.2.11
isu_round_select_tow	Section 2.3.6.3.3.7
turret_tow_movement_on	Section 2.3.6.1.1.30
ammo_tow_test_start	Section 2.3.5.1.47

Table 2.3-152: ammo_tow_select_pushed Information.

2.3.5.1.39 ammo_tow_test_pushed

This routine starts the TOW missile test when the TOW Test button is pressed. In order for the TOW test to start, the TOW launcher must be on, the gunners primary sight must be in 12X magnification, the TOW missile must have been selected, and the weapon must not already be in the process of firing.

Calls	
Function	**Where Described**
weapons_trigger_status	Section 2.3.3.2.13
ammo_tow_test_start	Section 2.3.5.1.47

Table 2.3-153: ammo_tow_test_pushed Information.

2.3.5.1.40 ammo_missile1_pushed

When the Missile1 button is pushed, this routine causes the system to be set for firing a missile from the missile1 tube. If the missile2 tube was previous selected, any TOW wires are cut. The TOW launcher must be up, the gunners primary sight must be in 12X magnification, and the TOW missile must be selected. If these conditions are met, the missile1 tube is selected, the missile is loaded into the missile1 tube and the missile1 tube controls are activated.

Calls	
Function	**Where Described**
weapons_cut_any_tow_wires	Section 2.3.3.2.9
ammo_get_missile_loaded	Section 2.3.5.1.61
controls_missile2_off	Section 2.3.2
controls_missile1_on	Section 2.3.2
controls_missile1_flash	Section 2.3.2

Table 2.3-154: ammo_missile1_pushed Information.

2.3.5.1.41 ammo_missile2_pushed

When the Missile2 button is pushed, this routine causes the system to be set for firing a missile from the missile2 tube. If the missile1 tube was previous selected, any TOW wires are cut. The TOW launcher must be up, the gunners primary sight must be in 12X magnification, and the TOW missile must be selected. If these conditions are met, the missile1 tube is selected, the missile is loaded into the missile2 tube, and the missile2 tube controls are activated.

Calls	
Function	**Where Described**
weapons_cut_any_tow_wires	Section 2.3.3.2.9
ammo_get_missile_loaded	Section 2.3.5.1.61
controls_missile1_off	Section 2.3.2
controls_missile2_on	Section 2.3.2
controls_missile2_flash	Section 2.3.2

Table 2.3-155: ammo_missile2_pushed Information.

2.3.5.1.42 ammo_tow_launcher_on

This routine turns on the TOW launcher by setting *tow_launcher_val* to ON.

2.3.5.1.43 ammo_tow_launcher_off

This routine turns off the TOW launcher. *tow_launcher_val* is set to OFF. If the TOW missile is the selected round, the weapon is set to single shot mode, the TOW test is stopped, the ammo is unloaded, the missile1 and missile2 tubes are unselected, and any TOW wires are cut.

Calls	
Function	**Where Described**
controls_round_select_no_round	Section 2.3.2
weapons_set_single_shot_mode	Section 2.3.3.2.11
isu_round_select_no_round	Section 2.3.6.3.3.6
turret_tow_movement_off	Section 2.3.6.1.1.29
ammo_tow_test_stop	Section 2.3.5.1.48
weapons_cut_any_tow_wires	Section 2.3.3.2.9

Table 2.3-156: ammo_tow_launcher_off Information.

2.3.5.1.44 ammo_gps_mag_12x

This routine sets the gunners primary sight magnification value to 12X. If firing a TOW missile, the gps magnification must be 12x.

2.3.5.1.45 ammo_gps_mag_4x

This routine sets the gunners primary sight magnification value to 4x. A TOW missile may not be fired with gps magnification of 4x. If the TOW missile was selected, the weapon is set to single shot mode, the TOW test is stopped, the ammo is unloaded, the missile1 and missile2 tubes are unselected, and any TOW wires are cut.

Calls	
Function	**Where Described**
controls_round_select_no_round	Section 2.3.2
weapons_set_single_shot_mode	Section 2.3.3.2.11
isu_round_select_no_round	Section 2.3.6.3.3.6
turret_tow_movement_off	Section 2.3.6.1.1.29
ammo_tow_test_stop	Section 2.3.5.1.48
weapons_cut_any_tow_wires	Section 2.3.3.2.9

Table 2.3-157: ammo_gps_mag_4x Information.

2.3.5.1.46 ammo_tow_test_check

On a tick by tick basis, this routine is called to check whether a TOW test is in progress. The TOW test is a test of the TOW missile electrical circuits. Controls are notified to turn on the TOW Test indicator when the TOW test is in progress, and turn off the indicator when the TOW test is not in progress.

Calls	
Function	**Where Described**
electsys_tow_circuit_open_status	Section 2.3.6.3.1.16
controls_tow_test_on	Section 2.3.2
controls_tow_test_off	Section 2.3.2

Table 2.3-158: ammo_tow_test_check Information.

2.3.5.1.47 ammo_tow_test_start

This routine sets the tow_test_in_progress flag to TRUE.

2.3.5.1.48 ammo_tow_test_stop

This routine sets the tow_test_in_progress flag to FALSE.

2.3.5.1.49 ammo_low_ammo_check

This routine checks to see if there are sufficient rounds of the selected ammunition round. If the quantity of selected ammunition is equal to or lower than the amount set in N_LOW_AMMO, the *low_ammo_val* is set to TRUE. If the Low Ammo button was not pressed, the Low Ammo indicator flashes. If the Low Ammo indicator was acknowledged by pressing the Low Ammo button, the Low Ammo indicator turns on steady. If there are sufficient rounds of the selected ammo, the Low Ammo indicator is turned off.

Calls	
Function	**Where Described**
controls_low_ammo_flash	Section 2.3.2
controls_low_ammo_on	Section 2.3.2
controls_low_ammo_off	Section 2.3.2

Table 2.3-159: ammo_tow_test_check Information.

2.3.5.1.50 ammo_low_ammo_pushed

When the Low Ammo button is pressed and there are not sufficient rounds of the selected ammop, the *low_ammo_lock_val* is set to OFF.

2.3.5.1.51 ammo_low_ammo_ready_to_fire

This routine returns FALSE if the vehicle is low on the selected ammo and the Low Ammo button has not been pressed in acknowledgment. TRUE is returned if either the vehicle is not low on the selected ammo or the Low Ammo button has been pressed in acknowledgment.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	Either the ammo is not low or the Low Ammo button has been pressed.
FALSE	int	The ammo is low and the Low Ammo button has not been pressed.

Table 2.3-160: ammo_low_ammo_ready_to_fire Information.

2.3.5.1.52 ammo_turret_power_off

This routine is called when turret power is turned off. The weapon is placed in single shot mode, any rounds are unselected and unloaded, any Tow test is stopped, and any TOW wires are cut.

Calls	
Function	**Where Described**
controls_round_select_no_round	Section 2.3.2
weapons_set_single_shot_mode	Section 2.3.3.2.11
isu_round_select_no_round	Section 2.3.6.3.3.6
turret_tow_movement_off	Section 2.3.6.1.1.29
ammo_tow_test_stop	Section 2.3.5.1.48
weapons_cut_any_tow_wires	Section 2.3.3.2.9

Table 2.3-161: ammo_turret_power_off Information.

2.3.5.1.53 ammo_bolt_position_status

This routine returns the bolt position.

Return Values		
Return Value	**Type**	**Meaning**
bolt_position	int	The position of the bolt.

Table 2.3-162: ammo_bolt_position_status Information.

2.3.5.1.54 ammo_misfire_lock_status

This routine returns the value of *ammo_misfire_lock* (either OFF or ON).

Return Values		
Return Value	**Type**	**Meaning**
ammo_misfire_lock	int	If equal to OFF or FALSE, the misfire button has not been pressed; If equal to ON or TRUE, the misfire button has been pressed.

Table 2.3-163: ammo_misfire_lock_status Information.

2.3.5.1.55 ammo_weapon_removed

This routine removes the selected ammo from the appropriate ammo box and loads the weapon.

Errors	
Error	**Reason for Error**
stderr AMMO: ammo_weapon_removed	Impossible ammo round selected
Calls	
Function	**Where Described**
firectl_weapon_removed	Section 2.3.2.2.3
ammo_remove_hei_can_round	Section 2.3.5.1.65
ammo_remove_apds_can_round	Section 2.3.5.1.64
ammo_get_hei_can_first_round	Section 2.3.5.1.63
ammo_get_apds_can_first_round	Section 2.3.5.1.62
controls_missile1_flash	Section 2.3.2
need_to_send_veh_status	Section 2.1.1.3.1.32.1
controls_missile2_flash	Section 2.3.2
ammo_tow_test_stop	Section 2.3.2.1.48

Table 2.3-164: ammo_weapon_removed Information.

2.3.5.1.56 ammo_weapon_is_fired

This routine removes the selected ammo from the ammo box, loads the weapon, fires the weapon, and resets the bolt position to sear.

Calls	
Function	**Where Described**
ammo_weapon_removed	Section 2.3.5.1.55
controls_bolt_position_sear	Section 2.3.2

Table 2.3-165: ammo_weapon_is_fired Information.

2.3.5.1.57 ammo_weapon_is_misfired

This routine sets up the weapon misfire failure. The selected ammo is removed from the ammo box, the weapon is loaded, and the bolt position is set to misfire.

Calls	
Function	**Where Described**
ammo_weapon_removed	Section 2.3.5.1.55
controls_bolt_position_misfire	Section 2.3.2

Table 2.3-166: ammo_weapon_is_misfired Information.

2.3.5.1.58 ammo_misfire_corrected

This routine repairs the weapon misfire failure by resetting the bolt position to sear.

Calls	
Function	**Where Described**
controls_bolt_position_sear	Section 2.3.2

Table 2.3-167: ammo_misfire_corrected Information.

2.3.5.1.59 ammo_misfire_pushed

If the bolt is in misfire position and the Misfire button is pressed, the sound of the misfire button is made and *ammo_misfire_lock* is reset to FALSE.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.3-168: ammo_misfire_pushed Information.

2.3.5.1.60 ammo_restore_ammo

This routine resets the ammo system, filling up the vehicle with ammo. This routine is called in order to bypass the resupply process during debugging.

Calls	
Function	**Where Described**
controls_round_select_no_round	Section 2.3.2
weapons_set_single_shot_mode	Section 2.3.3.2.11
isu_round_select_not_round	Section 2.3.6.3.3.6
turret_tow_movement_off	Section 2.3.6.1.1.29
ammo_tow_test_stop	Section 2.3.5.1.48
controls_bolt_position_sear	Section 2.3.2
weapons_cut_any_tow_wires	Section 2.3.3.2.9
ammo_init_ammo_supply	Section 2.3.5.1.3
ammo_stop_resupply	Section 2.3.5.1.76

Table 2.3-169: ammo_restore_ammo Information.

2.3.5.1.61 ammo_get_missile_loaded

This routine returns *munition_US_TOW* if the loaded status is TRUE, and *EMPTY* if the loaded status is FALSE.

Return Values		
Return Value	**Type**	**Meaning**
munition_US_TOW	ObjectType	A TOW missile is loaded.
EMPTY	ObjectType	No missile is loaded.

Table 2.3-170: ammo_get_missile_loaded Information.

2.3.5.1.62 ammo_get_apds_can_first_round

This routine checks to see what type of ammo is contained in the apds ammo box. If the box contains apds, then *munition_US_M791* is returned. If the box contains hei, then *munition_US_M792* is returned. If the apds quantity is equal to zero, EMPTY is returned.

Return Values		
Return Value	**Type**	**Meaning**
EMPTY	ObjectType	There are no apds canisters in the vehicle.
munition_US_M791	ObjectType	The first round in the apds box is an apds round.
munition_US_M792	ObjectType	The first round in the apds box is an hei round.

Calls	
Function	**Where Described**
ammo_get_apds_can_box	Section

Table 2.3-171: ammo_get_apds_can_first_round Information.

2.3.5.1.63 ammo_get_hei_can_first_round

This routine checks to see what type of ammo is contained in the hei ammo box. If the box contains apds, then *munition_US_M791* is returned. If the box contains hei, then *munition_US_M792* is returned. If the hei quantity is equal to zero, EMPTY is returned.

Return Values		
Return Value	**Type**	**Meaning**
EMPTY	ObjectType	There are no hei canisters in the vehicle.
munition_US_M791	ObjectType	The first round in the hei box is an apds round.
munition_US_M792	ObjectType	The first round in the hei box is an hei round.

Calls	
Function	**Where Described**
ammo_get_hei_can_box	Section 2.3.5.1.67

Table 2.3-172: ammo_get_hei_can_first_round Information.

2.3.5.1.64 ammo_remove_apds_can_round

This routine decrements the quantity of apds canisters in the vehicle. If the quantity of apds canisters is zero, the routine does nothing.

Calls	
Function	**Where Described**
need_to_send_veh_status	Section 2.1.1.3.1.32.1

Table 2.3-173: ammo_remove_apds_can_round Information.

2.3.5.1.65 ammo_remove_hei_can_round

This routine decrements the quantity of hei canisters in the vehicle. If the quantity of hei canisters is zero, the routine does nothing.

Calls	
Function	**Where Described**
need_to_send_veh_status	Section 2.1.1.3.1.32.1

Table 2.3-174: ammo_remove_hei_can_round Information.

2.3.5.1.66 ammo_get_apds_can_box

This routine performs an existence check on the ammo box passed in *box_num*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
box_num	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
(apds_can_ammo_boxes >> box_num) & 0x1	char	The munition type in the ammo box.

Table 2.3-175: ammo_get_apds_can_box Information.

2.3.5.1.67 ammo_get_hei_can_box

This routine performs an existence check on the ammo box passed in *box_num*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
box_num	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
(hei_can_ammo_boxes >> box_num) & 0x1	char	The munition type in the ammo box.

Table 2.3-176: ammo_get_hei_can_box Information.

2.3.5.1.68 ammo_print_ammo_variables

This routine prints the different ammunition quantities for debugging.

2.3.5.1.69 ammo_ready_to_internal_resupply

This routine sets up the ammo system for an internal resupply. The turret must be in the appropriate load position, the Turret Travel Lock Switch must be LOCKED, the Turret Drive System Switch must be OFF, and the Internal pushbutton on the Ammunition Management Panel must be pressed. If any of these conditions are not met, the routine returns FALSE. The routine checks that there is room in the appropriate ammo box or missile tube, and that there is enough supply in stowage for the transfer. If so, the routine returns TRUE. If not, the routine returns FALSE.

Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	The system is not ready for an internal transfer.
ammo_hei_can_enough_room() && ammo_hei_stowage_enough_supply()	int	If TRUE, the system is ready for an internal transfer: there is room in the hei ammo box and sufficient hei in stowage; If FALSE, the system is not ready for an internal transfer
ammo_apds_can_enough_room() && ammo_hei_stowage_enough_supply()	int	If TRUE, the system is ready for an internal transfer: there is room in the apds ammo box and sufficient hei in stowage; If FALSE, the system is not ready for an internal transfer
ammo_hei_can_enough_room() && ammo_apds_stowage_enough_supply()	int	If TRUE, the system is ready for an internal transfer: there is room in the hei ammo box and sufficient apds in stowage; If FALSE, the system is not ready for an internal transfer
ammo_apds_can_enough_room() && ammo_apds_stowage_enough_supply()	int	If TRUE, the system is ready for an internal transfer: there is room in the apds ammo box and sufficient apds in stowage; If FALSE, the system is not ready for an internal transfer
ammo_tow_tubes_enough_room() && ammo_tow_stowage_enough_supply()	int	If TRUE, the system is ready for an internal transfer: there is room in the missile tubes and sufficient TOW missiles in stowage; If FALSE, the system is not ready for an internal transfer

Calls	
Function	**Where Described**
controls_turret_drive_system_status	Section 2.3.2
alpha_get_load	Section 2.3.2.3.1
ammo_hei_can_enough_room	Section 2.3.5.1.77
ammo_hei_stowage_enough_supply	Section 2.3.5.1.83
ammo_apds_can_enough_room	Section 2.3.5.1.78
ammo_apds_stowage_enough_supply	Section 2.3.5.1.84
ammo_tow_tubes_enough_room	Section 2.3.5.1.80
ammo_tow_stowage_enough_supply	Section 2.3.5.1.85
launcher_get_val	Section 2.3.6.1.4.6

Table 2.3-177: ammo_ready_to_internal_resupply Information.

2.3.5.1.70 ammo_ready_to_external_resupply

This routine sets up the ammo system for an external resupply. The turret must be in the appropriate load position, the Turret Travel Lock Switch must be LOCKED, the Turret Drive Ssytem Switch must be OFF, and the Receive pushbutton on the Ammunition Management Panel must be pressed. If any of these conditions are not met, the routine returns FALSE. The routine checks that there is room in either the appropriate ammo box, missile tube, or stowage for the receipt of ammo. If so, the routine returns TRUE. If not, the routine returns FALSE.

Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	The system is not ready for an external resupply.
ammo_hei_can_enough_ room() || ammo_25mm_ stowage_enough_room()	int	If TRUE, the system is ready for an external resupply: there is either room in the ammo box or in stowage for hei; If FALSE, the system is not ready for an external resupply.
ammo_apds_can_enough_ room() || amm_25mm_ stowage_enough_room()	int	If TRUE, the system is ready for an external resupply: there is either room in the ammo box or in stowage for apds; If FALSE, the system is not ready for an external resupply.
ammo_25mm_stowage_ enough_room()	int	If TRUE, the system is ready for an external resupply: there is room in stowage; If FALSE, the system is not ready for an external resupply.
ammo_tow_tubes_enough_ room() || ammo_tow_ stowage_enough_room()	int	If TRUE, the system is ready for an external resupply: there is either room in the missile tubes or in stowage for TOW missiles; If FALSE, the system is not ready for an external resupply.
ammo_tow_stowage_ enough_room()	int	If TRUE, the system is ready for an external resupply: there is room in stowage; If FALSE, the system is not ready for an external resupply.

ammo_dragon_stowage_ enough_room()	int	If TRUE, the system is ready for an external resupply: there is room in stowage for dragon missiles; If FALSE, the system is not ready for an external resupply.

Calls	
Function	**Where Described**
controls_turret_drive_system_status	Section 2.3.2
alpha_get_load	Section 2.3.2.3.1
ammo_hei_can_enough_room	Section 2.3.5.1.77
ammo_25mm_stowage_enough_room	Section 2.3.5.1.79
ammo_apds_can_enough_room	Section 2.3.5.1.78
ammo_tow_tubes_enough_room	Section 2.3.5.1.80
ammo_tow_stowage_enough_room	Section 2.3.5.1.81
ammo_dragon_stowage_enough_room	Section 2.3.5.1.82

Table 2.3-178: ammo_ready_to_external_resupply Information.

2.3.5.1.71 ammo_ready_to_external_send

This routine sets up the ammo system for an external ammo send. If the Send pushbutton on the Ammunition Management Panel is pressed, the turret drive system is on, and there is sufficient ammo in stowage to transfer, the routine returns TRUE. If any of these conditions are not met, the routine returns FALSE.

Return Values		
Return Value	**Type**	**Meaning**
(ammo_transfer_status == AMMO_SEND_VAL) && (! controls_turret_drive_ system_status()) && (! ammo_supply_empty_ stowage())	int	If TRUE, the vehicle is ready for an external send. If FALSE, the vehicle is not ready for an external send.

Calls	
Function	**Where Described**
controls_turret_drive_system_status	Section 2.3.2
ammo_supply_empty_stowage	Section 2.3.5.1.72

Table 2.3-179: ammo_ready_to_external_send Information.

2.3.5.1.72 ammo_supply_empty_stowage

This routine no ammunition is left in stowage, this routine returns TRUE.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	There is no ammo in stowage.
FALSE	int	There is ammo in stowage.

Table 2.3-180: ammo_supply_empty_stowage Information.

2.3.5.1.73 ammo_start_internal_resupply

This routine starts the internal resupply. The routine notifies the controls to flash the selected ammo pushbutton lamp for the amount of time in either the DELAY_25MM constant or the DELAY_MISSILE constant, depending upon which type of ammo is selected. The *internal_resupply_in_progress* variable is set to TRUE.

Calls	
Function	**Where Described**
controls_internal_flash	Section 2.3.2
controls_hei_flash	Section 2.3.2
controls_apds_flash	Section 2.3.2
controls_tow_flash	Section 2.3.2

Table 2.3-181: ammo_start_internal_resupply Information.

2.3.5.1.74 ammo_start_external_resupply

This routine starts the external resupply. *apds25*, *hei25*, *tow*, and *dragon* are the quantities of specific ammo types available for transfer from the sending vehicle.

The routine notifies the controls to flash both the Receive pushbutton lamp and the selected ammo pushbutton lamp for the amount of time in either the DELAY_25MM constant or the DELAY_MISSILE constant. If the external resupply was successfully started, the routine returns TRUE. If no ammunition of the desired type is available from the sending vehicle, the routine returns FALSE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
apds25	int	Standard
hei25	int	Standard
tow	int	Standard
dragon	int	Standard

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	Resupply of the selected ammo has started.
FALSE	int	Resupply of the selected ammo is not possible.

Calls	
Function	**Where Described**
controls_receive_flash	Section 2.3.2
controls_hei_flash	Section 2.3.2
controls_apds_flash	Section 2.3.2
controls_tow_flash	Section 2.3.2
controls_dragon_flash	Section 2.3.2

Table 2.3-182: ammo_start_external_resupply Information.

2.3.5.1.75 ammo_start_external_send

This routine starts the external send. Controls are notified to flash the Send indicator.

Calls	
Function	**Where Described**
controls_send_flash	Section 2.3.2

Table 2.3-183: ammo_start_external_send Information.

2.3.5.1.76 ammo_stop_resupply

This routine stops a resupply, resetting all indicators.

Calls	
Function	**Where Described**
resupply_stop_ammo_resupply	Section 2.3.5.3.30
controls_receive_on	Section 2.3.2
controls_send_off	Section 2.3.2
controls_internal_off	Section 2.3.2
controls_hei_on	Section 2.3.2
controls_hei_off	Section 2.3.2
controls_apds_on	Section 2.3.2
controls_apds_off	Section 2.3.2
controls_tow_on	Section 2.3.2
controls_tow_off	Section 2.3.2
controls_dragon_on	Section 2.3.2
controls_dragon_off	Section 2.3.2

Table 2.3-184: ammo_stop_resupply Information.

2.3.5.1.77 ammo_hei_can_enough_room

This routine determines if there is room for more hei ammo in the vehicle.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	There is room for more hei in the vehicle.
FALSE	int	There is no more room for hei in the vehicle.

Table 2.3-185: ammo_hei_can_enough_room Information.

2.3.5.1.78 ammo_apds_can_enough_room

This routine determines if there is room for more apds ammo in the vehicle.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	There is room for more apds in the vehicle.
FALSE	int	There is no more room for apds in the vehicle.

Table 2.3-186: ammo_apds_can_enough_room Information.

2.3.5.1.79 ammo_25mm_stowage_enough_room

This routine determines if there is room for more 25mm ammo (either hei or apds) in stowage.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	There is room for more 25mm ammo in stowage.
FALSE	int	There is no more room for 25mm ammo in stowage.

Table 2.3-187: ammo_25mm_stowage_enough_room Information.

2.3.5.1.80 ammo_tow_tubes_enough_room

This routine determines if there is room for more missiles in the TOW missile tubes.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	There is room for more missiles in the missile tubes.
FALSE	int	Both missile tubes are full.

Table 2.3-188: ammo_tow_tubes_enough_room Information.

2.3.5.1.81 ammo_tow_stowage_enough_room

This routine determines if there is room for more TOW missiles in stowage.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	There is room for more TOW missiles in stowage.
FALSE	int	There is no room for more TOW missiles in stowage.

Table 2.3-189: ammo_tow_stowage_enough_room Information.

2.3.5.1.82 ammo_dragon_stowage_enough_room

This routine determines if there is room for more dragon missiles in stowage.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	There is room for more dragon missiles in stowage.
FALSE	int	There is no room for more dragon missiles in stowage.

Table 2.3-190: ammo_dragon_stowage_enough_room Information.

2.3.5.1.83 ammo_hei_stowage_enough_supply

This routine determines if there is a sufficient supply of hei ammo in stowage.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	There is a sufficient supply of hei.
FALSE	int	There is not a sufficient supply of hei.

Table 2.3-191: ammo_hei_hei_stowage_supply Information.

2.3.5.1.84 ammo_apds_stowage_enough_supply

This routine determines if there is a sufficient supply of apds ammo in stowage.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	There is a sufficient supply of apds.
FALSE	int	There is not a sufficient supply of apds.

Table 2.3-192: ammo_apds_stowage_enough_supply Information.

2.3.5.1.85 ammo_tow_stowage_enough_supply

This routine determines if there is a sufficient supply of TOW missiles in stowage.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	There is at least one TOW missile in stowage.
FALSE	int	There are no TOW missiles in stowage.

Table 2.3-193: ammo_tow_stowage_enough_supply Information.

2.3.5.1.86 ammo_turret_no_power_off

This routine stops any internal resupply in progress, sets the management round to empty, and resets the resupply counter to zero.

2.3.5.1.87 ammo_internal_resupply_start_check

On a tick by tick basis, this routine is called to see if the conditions for starting an internal resupply are met. In order to perform an internal resupply, the following conditions must be met: the resupply counter is set at zero, no internal transfer is currently in progress, the Internal pushbutton has been pressed, a valid round has been selected for transfer, and the resupply gating conditions have been met. If the conditions are met, the internal resupply is started.

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.3.5.3.15
ammo_ready_to_internal_resupply	Section 2.3.5.1.69
ammo_start_internal_resupply	Section 2.3.5.1.73

Table 2.3-194: ammo_internal_resupply_start_check Information.

2.3.5.1.88 ammo_internal_resupply_abort_check

On a tick by tick basis, this routine is called to see if any of the resupply abort conditions have been met. An internal resupply which was in progress is aborted if either the resupply gating conditions are no longer met or the **ammo_ready_to_internal_resupply** routine returns FALSE.

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.3.5.3.15
ammo_ready_to_internal_resupply	Section 2.3.5.1.69
ammo_stop_resupply	Section 2.3.5.1.76

Table 2.3-195: ammo_internal_resupply_abort_check Information.

2.3.5.1.89 ammo_resupply_timeout_check

This routine is called on a tick by tick basis to check for an ammo resupply timeout.

For an external resupply, the routine checks to see if the selected ammo type was received. If so, the inventory quantities for that type of ammo are adjusted and a vehicle status message is sent.

The routine does nothing in the case of an external send.

For an internal resupply, the routine adjusts the inventory quantities for the new ammo locations and sends a vehicle status message.

Calls	
Function	**Where Described**
controls_hei_on	Section 2.3.2
controls_receive_on	Section 2.3.2
resupply_ammo_received	Section 2.3.5.3.10
alpha_get_load	Section 2.3.2.3.1
ammo_hei_can_enough_room	Section 2.3.5.1.77
need_to_send_veh_status	Section 2.1.1.3.1.32.1
ammo_apds_can_enough_room	Section 2.3.5.1.78
controls_apds_on	Section 2.3.2
controls_tow_on	Section 2.3.2
controls_missile1_on	Section 2.3.2
controls_missile2_on	Section 2.3.2
controls_dragon_on	Section 2.3.2
controls_internal_on	Section 2.3.2

Table 2.3-196: ammo_internal_resupply_start_check Information.

2.3.5.1.90 ammo_rounds_on_board_check

On a tick by tick basis, this routine is called to update the Rounds on Board counter for each ammo type.

Calls	
Function	**Where Described**
controls_rounds_on_board	Section 2.3.2
controls_blank_rounds_on_board	Section 2.3.2

Table 2.3-197: ammo_rounds_on_board_check Information.

2.3.5.1.91 ammo_resupply_sent

This routine stops the external send and recalculates the stowage inventory for the type of ammo sent. *ammo_type* is the ammunition type that was sent.

Calls	
Function	**Where Described**
need_to_send_veh_status	Section 2.1.1.3.1.32.1
ammo_stop_resupply	Section 2.3.5.1.76

Table 2.3-198: ammo_rounds_on_board_check Information.

2.3.5.1.92 ammo_decide_round_type

This routine returns the selected round.

Return Values		
Return Value	**Type**	**Meaning**
ammo_mgmt_round	ObjectType	The selected round.

Table 2.3-199: ammo_decide_round_type Information.

2.3.5.2 m2_fuelsys.c

(./simnet/release/src/vehicle/m2/src/m2_fuelsys.c [m2_fuelsys.c])

The M2's fuel supply is monitored by this CSU. The M2 uses diesel fuel which is stored in two separate but interconnected fuel tanks. The top tank has approximately a 30 gallon capacity and the lower tank has approximately a 145 gallon capacity, for a total capacity of 175 gallons. The engine runs off the upper tank while the fuel gauge indicates the quantity in the lower tank. The fuel control handle starts and stops fuel flow to the engine. If the engine accessory is off, fuel is not transferred from the bottom tank to the top tank, thus, the engine can only use as much fuel as is held by the top tank. The fuel low light is not modeled. Variables for resupply: 30 gallons per minute times 1 min/60 sec times 1 sec/15 frames = 0.0333333333333333 gallons per frame.

This file includes:

- "stdio.h"
- "math.h"
- "sim_dfns.h"
- "sim_types.h"
- "sim_macros.h"
- "libfail.h"
- "failure.h"
- "m3_fuelsys.h"
- "m2_fuel_df.h"
- "m2_meter.h"
- "timers.h"
- "timers_dfn.h"
- "m2_resupp.h"
- "m2_sound.h"

Declarations:

fuel_xfer_fuel()	
fuel_top_tank_not_empty()	
fuel_resupply_tank()	
fuelsys_status	-- The status of the fuel button
fuel_flow	
top_fuel_level	
bottom_fuel_level	
engine_accessory	
resupply_timer_id	-- The timer id for resupply
tank_being_resupplied	-- The tank which is receiving fuel
total_resupplied	-- The fuel taken in a resupply interval
resupply_status	-- If TRUE, a resupply is in progress
mcc_offering	-- The amount of fuel offered by the MCC

Defines:

RESUPPLY_INTERVAL	-- Interval (seconds) before sending reply to the MCC
RESUPPLY_RATE	-- GPM of fuel transfer from truck
FUEL_PER_INTERVAL	

2.3.5.2.1 fuel_init_tanks

This routine is used by the MCC to intitialize the fuel level in each tank. This routine should be called before **fuel_init()**.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
top	REAL	sim_types.h
bottom	REAL	sim_types.h

Table 2.3-200: fuel_init_tanks Information.

2.3.5.2.2 fuel_init

This routine initializes the fuel system. The resupply status variables and failures are initialized.

Calls	
Function	**Where Described**
fail_init_failure	Section 2.5.4.11.2

Table 2.3-201: fuel_init Information.

2.3.5.2.3 fuel_simul

This routine simulates the various functions of the fuel system on a tick by tick basis. These functions consist of: 1) monitoring the fuel levels in each tank, 2) transferring fuel between the top and bottom tanks, 3) resupplying fuel, 4) stopping the fuel resupply, and 5) setting the meter amount of fuel in the bottom tank. Note that fuel may be transferred between the top and bottom tanks when the engine accessory is on and the engine is running.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
fuel_usage_this_tick	REAL	sim_types.h

Calls	
Function	**Where Described**
engine_running	Section 2.3.6.2.5.10
electsys_fuel_xfer_pump_request	Section 2.3.6.3.1.25
fuel_xfer_fuel	Section 2.3.5.2.6
fuel_resupply_tank	Section 2.3.5.2.15
timers_get_ticking_status	Section 2.6.3.20.1
fuel_stop_resupply	Section 2.3.5.2.14
meter_fuel_set	Section 2.3.2.3.3

Table 2.3-202: fuel_simul Information.

2.3.5.2.4 fuel_top_tank_not_empty

This routine returns TRUE if the top fuel tank is not empty and FALSE if the top fuel tank is empty.

Return Values		
Return Value	**Type**	**Meaning**
top_fuel_level > 0.0	BOOLEAN	If TRUE, the top fuel tank is not empty; if FALSE, the top fuel tank is empty

Table 2.3-203: fuel_top_tank_not_empty Information.

2.3.5.2.5 fuel_set_flow

This routine sets the *fuel_flow* variable equal to the parameter *value*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
value	REAL	sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
FALSE	BOOLEAN	Either the fuel button is OFF, the top fuel tank is empty, or an error has occurred
fuelsys_status && fuel_top_tank_not_empty()	BOOLEAN	If TRUE, then the fuel button is ON, the top fuel tank is not empty, and the *fuel_flow* variable is set to *value*; If FALSE, then either the fuel button is OFF or the top fuel tank is empty

Errors	
Error	**Reason for Error**
PANIC	negative fuel flow rate

Calls	
Function	**Where Described**
fuel_top_tank_not_empty	Section 2.3.5.2.4

Table 2.3-204: fuel_set_flow Information.

2.3.5.2.6 fuel_xfer_fuel

This routine calculates the top and bottom fuel tank levels after a fuel transfer between tanks.

2.3.5.2.7 fuel_engine_accessory_on

This routine is called by the controls software to let the fuel system know that the engine accessory has been turned on.

Calls	
Function	**Where Described**
sound_of_engine_accessory_on	Section 2.1.3.3.25

Table 2.3-205: fuel_engine_accessory_on Information.

2.3.5.2.8 fuel_engine_accessory_off

This routine is called by the controls software to let the fuel system know that the engine accessory has been turned off. When the engine accessory is turned off, fuel can no longer be transferred.

Calls	
Function	**Where Described**
sound_of_engine_accessory_off	Section 2.1.3.3.27

Table 2.3-206: fuel_engine_accessory_off Information.

2.3.5.2.9 fuel_level_bottom

This routine returns the fuel level of the bottom fuel tank.

Return Values		
Return Value	**Type**	**Meaning**
bottom_fuel_level	REAL	the level in the bottom fuel tank

Table 2.3-207: fuel_level_bottom Information.

2.3.5.2.10 fuel_level_top

This routine returns the fuel level of the top fuel tank.

Return Values		
Return Value	**Type**	**Meaning**
top_fuel_level	REAL	the level in the top fuel tank

Table 2.3-208: fuel_level_top Information.

2.3.5.2.11 fuel_supply_full

This routine calculates whether the amount of fuel in the parameter *delta* will fill the fuel tanks.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
delta	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	*delta* will fill the fuel supply
FALSE	BOOLEAN	*delta* will not fill the fuel supply

Table 2.3-209: fuel_supply_full Information.

2.3.5.2.12 fuel_decide_resupply_quantity

This routine calculates the amount of fuel needed to fill both tanks and returns either the maximum quanity of fuel allowed to transfer in one resupply interval or the actual amount of fuel necessary to fill both tanks (whichever is lower).

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
fuel_needed	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
(int) min(fuel_needed, FUEL_PER_INTERVAL)	int	the quantity of fuel necessary for resupply

Table 2.3-210: fuel_decide_resupply_quantity Information.

2.3.5.2.13 fuel_start_external_resupply

This routine starts the external resupply of fuel process. If the amount of fuel the MCC is offering is less than 0 gallons, the routine returns FALSE. Otherwise, the routine starts the resupply timer, determines which tanks should be resupplied, and returns TRUE.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fuel_offered	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	the external resupply was started
FALSE	BOOLEAN	the external resupply was not started
Calls		
Function	**Where Described**	
timers_get_timers	Section 2.6.3.6.1	

Table 2.3-211: fuel_start_external_resupply Information.

2.3.5.2.14 fuel_stop_resupply

This routine stops the fuel resupply. The timers are freed, the resupply status is set to OFF, and the amount of fuel received during the resupply is calculated.

Calls	
Function	**Where Described**
timers_free_timers	Section 2.6.3.5.1
resupply_fuel_received	Section 2.3.5.3.11

Table 2.3-212: fuel_stop_resupply Information.

2.3.5.2.15 fuel_resupply_tank

This routine calculates the amount of fuel in each tank after a fuel resupply.

Errors	
Error	**Reason for Error**
stderr	WARNING: invalid tank selection for resupply

Table 2.3-213: fuel_resupply_tank Information.

2.3.5.2.16 print_fuel_variables

This routine prints the fuel variables: *top_fuel_level*, *bottom_fuel_level*, *engine_accessory*, and *fuelsys_status*.

2.3.5.2.17 fuel_on

This routine sets the *fuelsys_status* flag to ON.

2.3.5.2.18 fuel_off

This routine sets the *fuelsys_status* flag to OFF.

2.3.5.2.19 fuel_transfer_pump_failed

This routine was added for future expansion.

2.3.5.2.20 fuel_repair_transfer_pump

This routine was added for future expansion.

2.3.5.3 m2_resupp.c
(./simnet/release/src/vehicle/m2/src/m2_resupp.c [m2_resupp.c])

This CSU coordinates the resupply of ammunition and fuel to the M2 simulator. The M2 simulator may also resupply ammunition (but not fuel) to other M2 simulators.

This file includes:
"stdio.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"timers_dfn.h"
"timers.h"
"mun_type.h"
"libnetwork.h"
"pro_sim.h"
"m2_ammo_df.h"
"m2_turr_mx.h"
"m2_ammo.h"
"m3_cntrl.h"
"m2_dtrain.h"
"m2_repair.h"
"m2_fuel_df.h"
"m2_launcher.h"
"pro_assoc.h"
"assoc.h"

Defines:
QUIET
REQUEST
LOADING
WAITING
SERVICING
MAX_SERVICE_ENTITIES

Declarations:

ammo_carrier
fuel_carrier
ammo_receiver
num_ammo_carriers
num_fuel_carriers
num_ammo_receivers
ammo_carriers_near_here
fuel_carriers_near_here
ammo_receivers_near_here

ammo_resupply_receive_state
ammo_has_been_received
ammo_that_was_received
ammo_receive_timer_id
ammo_carrier_id

fuel_resupply_receive_stae
fuel_has_been_received
fuel_that_was_received
fuel_receive_timer_id
fuel_carrier_id
ammo_resupply_send_state
ammo_send_timer_id
ammo_receiver_id

clear_ammo_carriers()
clear_fuel_carriers()
clear_ammo_receivers()
send_feed_me_packets_ammo_carriers()
send_feed_me_packets_fuel_carriers()
ammo_receive_quiet_state()
fuel_receive_quiet_state()
ammo_send_quiet_state()
ammo_receive_request_state()
fuel_receive_request_state()
ammo_send_waiting_state()
ammo_receive_loading_state()
fuel_receive_loading_state()
ammo_send_servicing_state()
ammo_resupply_receive_simul()
fuel_resupply_receive_simul()
ammo_resupply_send_simul()
vehicle_is_close()

2.3.5.3.1 clear_ammo_carriers

This routine clears the *ammo_carriers_near_here* flag to FALSE and the number of ammo carriers to zero.

2.3.5.3.2 clear_fuel_carriers

This routine clears the *fuel_carriers_near_here* flag to FALSE and the number of fuel carriers to zero.

2.3.5.3.3 clear_ammo_receivers

This routine clears the *ammo_carriers_near_here* flag to FALSE and the number of ammo receivers to zero.

2.3.5.3.4 print_resupply_status

This routine prints the resupply status information.

2.3.5.3.5 send_feed_me_packets_ammo_carriers

This routine sends a service request packet to each of the ammo carriers on the network, requesting the specific types of ammunition necessary for resupply.

Internal Variables		
Variable	Type	Where Typedef Declared
i	int	Standard
munition	MunitionQuantity	basic.h

Calls	
Function	Where Described
ammo_decide_round_type	Section 2.3.5.1.92
network_send_feed_me_packet	Section 2.1.1.3.1.48.1

Table 2.3-214: send_feed_me_packets_ammo_carriers Information.

2.3.5.3.6 send_feed_me_packets_fuel_carriers

This routine sends a service request packet to each of the fuel carriers on the network, requesting a specific quantity of fuel necessary for resupply.

Internal Variables		
Variable	Type	Where Typedef Declared
i	int	Standard
munition	MunitionQuantity	basic.h

Calls	
Function	Where Described
fuel_decide_resupply_quantity	Section 2.3.5.2.12
network_send_feed_me_packet	Section 2.1.1.3.1.48.1

Table 2.3-215 send_feed_me_packets_fuel_carriers Information.

2.3.5.3.7 resupply_near_ammo_carrier

This routine maintains the list of close vehicles which are ammo carriers. If any ammo carriers are on this list, the *ammo_carriers_near_here* flag is set to TRUE.

Parameters		
Parameter	Type	Where Typedef Declared
v	pointer to VehicleID	basic.h

Table 2.3-216: resupply_near_ammo_carrier Information.

2.3.5.3.8 resupply_near_fuel_carrier

This routine maintains the list of close vehicles which are fuel carriers. If any fuel carriers are on this list, the *fuel_carriers_near_here* flag is set to TRUE.

Parameters		
Parameter	Type	Where Typedef Declared
v	pointer to VehicleID	basic.h

Table 2.3-217: resupply_near_fuel_carrier Information.

2.3.5.3.9 resupply_near_ammo_receiver

This routine maintains the list of close vehicles which are ammo receivers. If any ammo receivers are on this list, the *ammo_receivers_near_here* flag is set to TRUE.

Parameters		
Parameter	Type	Where Typedef Declared
v	pointer to VehicleID	basic.h

Table 2.3-218: resupply_near_ammo_receiver Information.

2.3.5.3.10 resupply_ammo_received

This routine sets the *ammo_has_been_received* flag to TRUE, and set the variable *ammo_that_was_received* to the quantities and types of ammunition that was received.

Parameters		
Parameter	Type	Where Typedef Declared
ammo_type	ObjectType	p_sim.h

Table 2.3-219: resupply_ammo_received Information.

2.3.5.3.11 resupply_fuel_received

This routine sets the *fuel_has_been_received* flag to TRUE, and set the variable *fuel_that_was_received* equal to the number of gallons of fuel that were received.

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

Table 2.3-220: resupply_fuel_received Information.

2.3.5.3.12 resupply_offer_packet

This routine is called by the LibRcvNet routine **process_resupply_offer()** in order to process a message from a vehicle within range offering munitions resupply. **resupply_offer_packet()** first determines the types and quantities of munitions carried, and then checks the ammo and fuel resupply receive states.

If the resupply receive state is QUIET, no munitions have been requested and no munitions are received. If the state is REQUEST, the resupply timer is started a, the external resupply of either fuel or ammunition (apds25, hei25, or tow) is started, and the state is changed to LOADING. If the state is LOADING, the external resupply is in progress. If the receive state is not known, print an error. Parameters are represented as follows:

carrier_id -- The VehicleID of the munitions carrier.
num_munitions -- The number of munitions types carried.
munitions -- The quantities and types of munitions being carried.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
carrier_id	pointer to VehicleID	basic.h
num_munitions	usigned char	Standard
munitions	register pointer to MunitionQuantity	basic.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
fuel	int	Standard
apds25	int	Standard
hei25	int	Standard
tow	int	Standard
dragon	int	Standard
mun_type	ObjectType	p_sim.h
mun_quantity	float	Standard
i	int	Standard

Errors	
Error	**Reason for Error**
RESUPPLY: resupply_offer_packet	- unkown ammo state - unknown fuel state
Calls	
Function	**Where Described**
timers_free_timer	Section 2.6.3.5.1
ammo_start_external_resupply	Section 2.3.5.1.74
timers_get_timer	Section 2.6.3.6.1

Table 2.3-221: resupply_offer_packet Information.

2.3.5.3.13 resupply_thank_you_packet

This routine is called by the LibRcvNet routine **process_resupply_received**() in order to process a message from a resupply receiver saying that the ammo was received. **resupply_thank_you_packet**() determines the types and quantities of ammo that were transferred, frees the resupply timer, sets the resupply send state to QUIET, and stops the ammo resupply.

receiver_id -- The Vehicle ID of the vehicle which received the ammo resupplies.
num_munitions -- The number of different types of ammo sent by the carrier.
munitions -- The quantity of each type of ammo sent by the carrier.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
receiver_id	pointer to VehicleID	basic.h
num_munitions	unsigned char	Standard
munitions	register pointer to MunitionQuantity	basic.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
i	int	Standard
Calls		
Function	**Where Described**	
ammo_resupply_send	Section 2.3.5.1.91	
timers_free_timers	Section 2.6.3.6.1	
ammo_stop_resupply	Section 2.3.5.1.76	

Table 2.3-222: resupply_thank_you_packet Information.

2.3.5.3.14 resupply_feed_me_packet

This routine is called by the LibRcvNet routine **process_service_request**() in order to process a message requesting ammunition resupply from a vehicle within range. **resupply_feed_me**() first checks the ammo resupply send state. If the state is QUIET, no supplies are to be sent. If the state is WAITING, send an offer packet on the network to the receiver listing the types and quantities of munitions that you have, start the external resupply, and change the state to SERVICING. If the state is SERVICING, the external resupply is in progress. If the state is not know, print an error.

receiver_id -- The VehicleID of the munitions receiver.
num_munitions -- The number of ammo types requested by the receiver.
feed_me_munitions -- The types and quantities of ammo being requested by the reciever.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
receiver_id	pointer to VehicleID	basic.h
num_munitions	unsigned char	Standard
feed_me_munitions	pointer to MunitionQuantity	basic.h

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
offer_munitions	pointer to MunitionQuantity	basic.h

Errors	
Error	**Reason for Error**
RESUPPLY: resupply_feed_me_packet	unknown ammo

Calls	
Function	**Where Described**
timers_free_timers	Section 2.6.3.5.1
timers_get_timers	Section 2.6.3.6.1
ammo_get_hei_stowed_quantity	Section 2.3.5.1.9
ammo_get_apds_stowed_quantity	Section 2.3.5.1.8
ammo_get_tow_stowed_quantity	Section 2.3.5.1.10
ammo_get_dragon_stowed_quantity	Section 2.3.5.1.11
network_send_offer_packet	Section 2.1.1.3.1.40.1
ammo_start_external_send	Section 2.3.5.1.75

Table 2.3-223: resupply_feed_me_packet Information.

2.3.5.3.15 resupply_gating_conditions

This routine returns TRUE if the vehicle is not moving and there are no failures in the controls. The routine returns FALSE if the vehicle is moving or there is a failure in the controls.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
tracks_speed	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	the vehicle is not moving; no failures in the controls
FALSE	BOOLEAN	either the vehicle is moving or there is a controls failure
Calls		
Function	**Where Described**	
drivetrain_get_vehicle_speed	Section 2.3.6.2.4.8	
controls_failure_status	Section 2.3.2	

Table 2.3-224: resupply_gating_conditions Information.

2.3.5.3.16 ammo_receive_quiet_state

This routine determines the receiver's ammunition resupply Finite State Machine's QUIET state. If the following conditions are ALL TRUE:

- The resupply gating conditions are TRUE (the vehicle is alive, the vehicle is not moving, and no controls failures exist).
- There are ammo carriers nearby.
- The receiver is ready to receive ammo (the commander has set the transfer mode to receive, the turret drive system is off, and there is room for the ammo).
- If TOW missiles are requested, the TOW launcher is up.

Then, send a feed me packet to the ammo carriers on the network, start the resupply timer, and enter the REQUEST state. If any of the conditions are not met, remain in the QUIET state.

Return Values		
Return Value	**Type**	**Meaning**
REQUEST	int	the receiver is in the REQUEST state
QUIET	int	the receiver is in the QUIET state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.3.5.3.15
ammo_ready_to_external_resupply	Section 2.3.5.1.70
ammo_decide_round_type	Section 2.3.5.1.92
launcher_get_val	Section 2.3.6.1.4.6
timers_get_timer	Section 2.6.3.6.1
send_feed_me_packets_ammo_carriers	Section 2.3.5.3.5

Table 2.3-225: ammo_receive_quiet_state Information.

2.3.5.3.17 fuel_receive_quiet_state

This routine determines the receiver's fuel resupply Finite State Machine's QUIET state. If the following conditions are ALL TRUE:

- The resupply gating conditions are TRUE (the vehicle is alive, the vehicle is not moving, and no controls failures exist).
- There are fuel carriers nearby.
- There is room for the fuel.

Then, send a feed me packet to the fuel carriers on the network, start the resupply timer, and enter the REQUEST state. If any of the conditions are not met, remain in the QUIET state.

Return Values		
Return Value	**Type**	**Meaning**
REQUEST	int	the receiver is in the REQUEST state
QUIET	int	the receiver is in the QUIET state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.3.5.3.15
fuel_supply_full	Section 2.3.5.2.11
timers_get_timer	Section 2.6.3.6.1
send_feed_me_packets_fuel_carriers	Section 2.3.5.3.6

Table 2.3-226: fuel_receive_quiet_state Information.

2.3.5.3.18 ammo_send_quiet_state

This routine determines the sender's ammunition resupply Finite State Machine's QUIET state. If the following conditions are ALL TRUE:

- The resupply gating conditions are TRUE (the vehicle is alive, the vehicle is not moving, and no controls failures exist).
- There are ammo receivers nearby.
- The ammo is ready to be sent.

Then, enter the WAITING state. If any of the conditions are not met, remain in the QUIET state.

Return Values		
Return Value	**Type**	**Meaning**
WAITING	int	the sender is in the WAITING state
QUIET	int	the sender is in the QUIET state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.3.5.3.15
ammo_ready_to_external_send	Section 2.3.5.1.72

Table 2.3-227 ammo_send_quiet_state Information.

2.3.5.3.19 ammo_receive_request_state

This routine determines the receiver's ammunition resupply Finite State Machine's REQUEST state. If ANY of the following conditions are TRUE:

- The resupply gating conditions are FALSE (the vehicle is dead, the vehicle is moving, or a controls failure exists).
- There are no ammo carriers nearby.
- The receiver is not ready to receive ammo (the commander has not set the transfer mode to receive, the turret drive system is on, or there is no room for the ammo).

Then, abort the resupply timer and enter the QUIET state. If none of the conditions are met and the resupply timer has expired, send another service request, restart the timer, and remain in the REQUEST state. If the resupply timer has not expired, remain in the REQUEST state.

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	the receiver is in the QUIET state
REQUEST	int	the receiver is in the REQUEST state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.3.5.3.15
ammo_ready_to_external_resupply	Section 2.3.5.1.70
timers_get_timeout_edge	Section 2.6.3.22.1
timers_free_timer	Section 2.6.3.5.1
timers_get_timer	Section 2.6.3.6.1
send_feed_me_packets_ammo_carriers	Section 2.3.5.3.6

Table 2.3-228: ammo_receive_request_state Information.

2.3.5.3.20 fuel_receive_request_state

This routine determines the receiver's fuel resupply Finite State Machine's REQUEST state. If ANY of the following conditions are TRUE:

- The resupply gating conditions are FALSE (the vehicle is dead, the vehicle is moving, or a controls failure exists).
- There are no fuel carriers nearby.
- The fuel supply is full.

Then, abort the resupply timer and enter the QUIET state. If none of the conditions are met and the resupply timer has expired, send another service request, restart the timer, and remain in the REQUEST state. If the resupply timer has not expired, remain in the REQUEST state.

Return Values		
Return Value	**Type**	**Meaning**
REQUEST	int	the receiver is in the REQUEST state
QUIET	int	the receiver is in the QUIET state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.3.5.3.15
fuel_supply_full	Section 2.3.5.2.11
timers_free_timers	Section 2.6.3.5.1
timers_get_timeout_edge	Section 2.6.3.22.1
timers_get_timer	Section 2.6.3.6.1
send_feed_me_packets_fuel_carriers	Section 2.3.5.3.6

Table 2.3-229: fuel_receive_request_state Information.

2.3.5.3.21 ammo_send_waiting_state

This routine determines the sender's ammunition resupply Finite State Machine's WAITING state. If ANY of the following conditions are TRUE:

- The resupply gating conditions are FALSE (the vehicle is dead, the vehicle is moving, or a controls failure exists).
- There are no ammo receivers nearby.
- The sender is not ready to send ammo (there are insufficient supplies to send).

Then, enter the QUIET state. If none of the conditions are met remain in the WAITING state.

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	the sender is in the QUIET state
WAITING	int	the sender is in the WAITING state

Calls	
Function	**Where Described**
resupply_gating_conditions	Section 2.3.5.3.15
ammo_ready_to_external_send	Section 2.3.5.1.71

Table 2.3-230: ammo_send_waiting_state Information.

2.3.5.3.22 ammo_receive_loading_state

This routine determines the receiver's ammunition resupply Finite State Machine's LOADING state. If the ammo has been received, a thank you packet is sent by the receiver listing the type and amount of ammunition taken, and the receiver enters the QUIET state.

If any of ANY of the following conditions have changed to TRUE in the LOADING state:

- The resupply gating conditions are FALSE (the vehicle is dead, the vehicle is moving, or a controls failure exists).
- There are no ammo carriers nearby.
- The ammo carrier which offered ammunition is dead.
- The receiver is no longer ready to receive ammo (the commander has not set the transfer mode to receive or the turret drive system is on).

Then, stop the resupply and enter the QUIET state.

If the loading has not completed, remain in the LOADING state.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
munitions	MunitionQuantity	basic.h

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	the sender is in the QUIET state
LOADING	int	the sender is in the LOADING state

Errors	
Error	**Reason for Error**
AMMO: ammo_receive_loading_state	impossible ammo_that_was_received

Calls	
Function	**Where Described**
network_send_thank_you_packet	Section 2.1.1.3.1.41.1
resupply_gating_conditions	Section 2.3.5.3.15
vehicle_is_close	Section 2.3.5.3.34
ammo_ready_to_external_resupply	Section 2.3.5.1.70
ammo_stop_resupply	Section 2.3.5.1.76

Table 2.3-231: ammo_receive_loading_state Information.

2.3.5.3.23 fuel_receive_loading_state

This routine determines the receiver's fuel resupply Finite State Machine's LOADING state. If the fuel has been received, a thank you packet is sent by the receiver listing the quantity of fuel taken, and the receiver enters the QUIET state.

If any of ANY of the following conditions have changed to TRUE in the LOADING state:

- The resupply gating conditions are FALSE (the vehicle is dead, the vehicle is moving, or a controls failure exists).
- There are no fuel carriers nearby.
- The fuel which offered fuel is dead.
- The fuel load is full.

Then, stop the resupply and send a thank you packet listing the quantity of fuel taken before the resupply was stopped and enter the QUIET state.

If the loading has not completed, remain in the LOADING state.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
munition	MunitionQuantity	basic.h

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	the sender is in the QUIET state
LOADING	int	the sender is in the LOADING state

Calls	
Function	**Where Described**
network_send_thank_you_packet	Section 2.1.1.3.1.41.1
resupply_gating_conditions	Section 2.3.5.3.15
fuel_supply_full	Section 2.3.5.2.11
vehicle_is_close	Section 2.3.5.3.34
fuel_stop_resupply	Section 2.3.5.2.14

Table 2.3-232: fuel_receive_loading_state Information.

2.3.5.3.24 ammo_send_servicing_state

This routine determines the sender's ammunition resupply Finite State Machine's SERVICING state. If the resupply timer has timed out, stop the resupply and enter the QUIET state. If the resupply timer has not timed out, remain in the SERVICING state.

Return Values		
Return Value	**Type**	**Meaning**
QUIET	int	the sender is in the QUIET state
SERVICING	int	the sender is in the SERVICING state

Calls	
Function	**Where Described**
timers_get_timeout_edge	Section 2.6.3.22.1
timers_free_timers	Section 2.6.3.5.1
ammo_stop_resupply	Section 2.3.5.1.76

Table 2.3-233: ammo_send_servicing_state Information.

2.3.5.3.25 ammo_resupply_receive_simul

This routine runs the ammunition resupply receive simulation. The routine checks the ammo resupply receive state and calls the appropriate routine for that state.

Errors	
Error	**Reason for Error**
REPAIR: ammo_resupply_receive_simul	unknown state

Calls	
Function	**Where Described**
ammo_receive_quiet_state	Section 2.3.5.3.16
ammo_receive_request_state	Section 2.3.5.3.19
ammo_receive_loading_state	Section 2.3.5.3.22

Table 2.3-234: ammo_resupply_receive_simul Information.

2.3.5.3.26 fuel_resupply_receive_simul

This routine runs the fuel resupply receive simulation. The routine checks the fuel resupply receive state and calls the appropriate routine for that state.

Errors	
Error	**Reason for Error**
REPAIR: fuel_resupply_receive_simul	unknown state
Calls	
Function	**Where Described**
fuel_receive_quiet_state	Section 2.3.5.3.17
fuel_receive_request_state	Section 2.3.5.3.20
fuel_receive_loading_state	Section 2.3.5.3.23

Table 2.3-235: fuel_resupply_receive_simul Information.

2.3.5.3.27 ammo_resupply_send_simul

This routine runs the ammunition resupply send simulation. The routine checks the sender's ammo resupply state and calls the appropriate routine for that state.

Errors	
Error	**Reason for Error**
REPAIR: ammo_resupply_send_simul	unknown state
Calls	
Function	**Where Described**
ammo_send_quiet_state	Section 2.3.5.3.18
ammo_send_waiting_state	Section 2.3.5.3.21
ammo_send_servicing_state	Section 2.3.5.3.24

Table 2.3-236: ammo_resupply_send_simul Information.

2.3.5.3.28 resupply_init

This routine initializes the resupply simulation. All ammo and fuel carriers and receivers are cleared, all resupply states are set to QUIET, and all resupply timers are set to NULL.

Calls	
Function	**Where Described**
clear_ammo_carriers	Section 2.3.5.3.1
clear_fuel_carriers	Section 2.3.5.3.2
clear_ammo_receivers	Section 2.3.5.3.3

Table 2.3-237: resupply_init Information.

2.3.5.3.29 resupply_simul

This routine runs the resupply simulations. The routine calls the ammo send, ammo receive simulation , and fuel receive routines.

Calls	
Function	**Where Described**
ammo_resupply_receive_simul	Section 2.3.5.3.25
fuel_resupply_receive_simul	Section 2.3.5.3.26
ammo_resupply_send_simul	Section 2.3.5.3.27
clear_ammo_carriers	Section 2.3.5.3.1
clear_fuel_carriers	Section 2.3.5.3.2
clear_ammo_receivers	Section 2.3.5.3.3

Table 2.3-238: resupply_simul Information.

2.3.5.3.29 service_check_vehicle_type

This routine checks the vehicle ID from the *pkt* parameter, determines its vehicle type, and updates the different lists of close vehicles (ammo carriers, fuel carriers, ammo receivers, etc.).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to VehicleAppearanceVariant	p_sim.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
id	pointer to VehicleID	basic.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	Successful

Calls	
Function	**Where Described**
is_fuel_vehicle	Section 2.6.10.7.1
resupply_near_fuel_carrier	Section 2.3.5.3.8
is_repair_vehicle	Section 2.6.10.10.1
repair_near_repair	Section 2.3.4.2.5
is_ammo_vehicle	Section 2.6.10.2.1
is_ammo_carrier	Section 2.6.10.2.2
resupply_near_ammo_carrier	Section 2.3.5.3.8
is_main_battle_tank	Section 2.6.10.9
resupply_near_ammo_receiver	Section 2.3.5.3.9

Table 2.3-239: service_check_vehicle_type Information.

2.3.5.3.30 resupply_stop_ammo_resupply

This routine aborts the ammo resupply simulation, resetting the ammo resupply send (or receive) state to QUIET and freeing the resupply timer.

Calls	
Function	**Where Described**
timers_free_timers	Section 2.6.3.5.1

Table 2.3-240: resupply_stop_ammo_resupply Information.

2.3.5.3.31 resupply_stop_fuel_resupply

This routine aborts the fuel resupply simulation, resetting the fuel resupply receive state to QUIET and freeing the resupply timer.

Calls	
Function	**Where Described**
timers_free_timers	Section 2.6.3.5.1

Table 2.3-241: resupply_stop_fuel_resupply Information.

2.3.5.3.32 resupply_offer_canceled

This routine cancels an offer of service from another vehicle.

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

Table 2.3-242: resupply_offer_canceled Information.

2.3.5.3.33 resupply_request_canceled

This routine cancels a request for service.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
receiver(id	int	Standard

Table 2.3-243: resupply_request_canceled Information.

2.3.5.3.34 vehicle_is_close

This routine determines if a particular vehicle is on the close vehicles list.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
list	register pointer to VehicleID	basic.h
vehicle	register pointer to VehicleID	basic.h
size_of_list	register int	Standard
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	The vehicle is on the close vehicles list
FALSE	int	The vehicle is not on the close vehicles list
Calls		
Function	**Where Described**	
VEHICLE_IDS_EQUAL	Macro defined in sim_macros.h	

Table 2.3-244: vehicle_is_close Information.

2.3.5.3.35 keybrd_ammo_carriers_near_here

This routine returns whether any ammo carriers are nearby, prompted by a user's keyboard request.

Return Values		
Return Value	**Type**	**Meaning**
ammo_carriers_near_here	BOOLEAN	Whether any ammo carriers are nearby

Table 2.3-245: keybrd_ammo_carriers_near_here Information.

2.3.6 M2 Vehicle Model

There are a number of vehicle specific simulation functions. Code is required for modeling the relevant moving elements of a vehicle. It is necessary to simulate the forces applied to the vehicle by its propulsion and suspension systems. The generation and use of electric and hydraulic power is simulated, as are the effects of the user's actions on the visual displays.

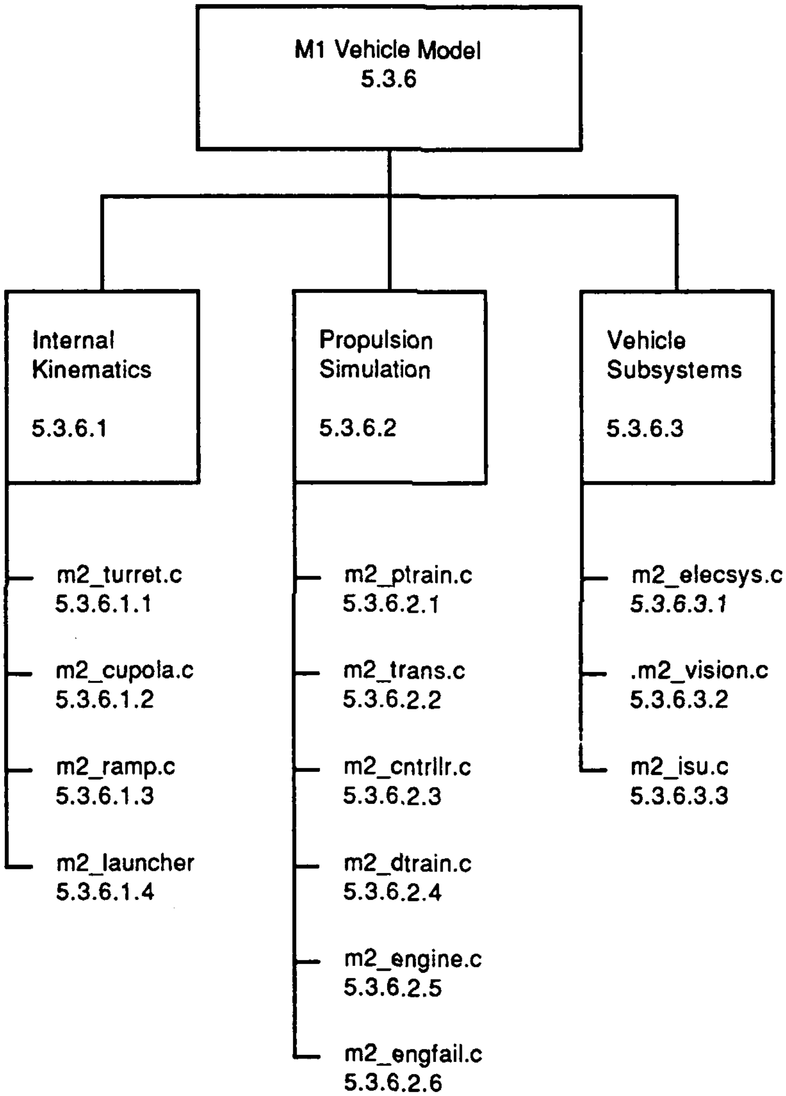

Figure 2.3-7: Structure of the M2 Vehicle Model CSC.

The third level CSC's are as follows:

Internal Kinematics
Propulsion Simulation
Vehicle Subsystems

2.3.6.1 Internal Kinematics

The M2 has components which move with respect to the hull. The turret and the TOW launcher on the M2 are examples. It is necessary to model the movement of these components; however, the required level of model fidelity varies among them. A much more detailed model is required for the turret than for the TOW launcher. The simulation of moving components is carried out by the following CSU's

m2_turret.c
m2_cupola.c
m2_ramp.c
m2_launcher.c

2.3.6.1.1 m2_turret.c
(./simnet/release/src/vehicle/m2/src/m2_turret.c [m2_turret.c])

The vehicle specific characteristics of the turret are modeled in m2_turret.c. The vehicle specific code responds to commands from the controls and determines the appropriate commands to send to libturret. The response depends on the mode of operation of the turret and is affected if a turret subsystem has failed. The operational modes and failure status are tracked in vehicle specific code. Commands are sent to the sound system from this module. This module contain functions which provide information to other components of the simulation, including the CIG and some of the control displays.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"timers.h"
"timers_dfn.h"
"pro_data.h"
"libturret.h"
"libmatrix.h"
"bigwheel.h"
"libkin.h"
"libfail.h"
"failure.h"
"libsound.h"
"m2_alpha.h"
"m2_bcs.h"
"m2_dtrain.h"
"m2_elecsys.h"
"m2_sound.h"
"m2_sound_dfn.h"
"m2_turr_def.h"
"m2_turret.h"
"m2_main.h"

Defines:

TURRET_DEBUG
TURRET_FAILURES_DEBUG
STAB_DEBUG
GUN_BREAK_SPEED

Declarations:

gyro_speed -- between 0.0 and 1.0
gun_slew_handle -- between -1.0 and 1.0
gun_elev_handle -- between -1.0 and 1.0
turret_slew_rate -- radians per frame
gun_elev_rate -- radians per frame
super_elev
sin_is_[illegible] -- radians of elevations (relative to the orientation of the hull)

sin_gun_elev	-- radians of elevations (relative to the orientation of the hull)
sin_tow_elev	-- radians of elevations (relative to the orientation of the hull)
sin_elev_rads	-- number of radians that the gun is elevated (relative to the orientation of the hull)
tow_movement	-- TRUE or FALSE
fast_movement	-- TRUE or FALSE
gun_on_stop	-- TRUE or FALSE
elevation_status	-- either WORKING or BROKEN
stab_status	-- either WORKING or BROKEN
mount_int_status	-- either WORKING or BROKEN
gearbox_status	-- either WORKING or BROKEN
traverse_status	-- either WORKING or BROKEN
stab_power	-- ON or OFF
control_engaged	-- ON or OFF

calc_elev_from_handle()
calc_slew_from_handle()
turret_gyros_simul()
turret_move()
turret_calc_gun_elev()
turret_calc_turret_slew()
make_sound_of_no_turret_noise()
make_sound_of_no_elevating()
make_sound_of_no_slewing()

2.3.6.1.1.1 turret_init

This routine initializes the turret variables. The stab vectors are also initialized in order to use the stabilization system.

Calls	
Function	**Where Described**
controls_turret_stab_off	Section 2.3.2
turret_set_stab_system	Section 2.5.5.2.3
fail_init_failure	Section 2.5.4.11.2

Table 2.3-246: turret_init Information.

2.3.6.1.1.2 turret_simul

This is the top level routine for the turret. This routine is called on a tick by tick basis to model the turret. Nothing will occur until the turret gyros are operational. When in manual mode, the stabilization is not operational. Note that the stab vectors must be set every tick, since they are set one tick ahead of use.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
old_control_value	int	Standard
turret_ref_ind	REAL	sim_types.h

Calls	
Function	**Where Described**
turret_gyros_simul	Section 2.3.6.1.1.11
sound_make_const_sound	Section 2.1.3.1.2
make_sound_of_no_turret_noise	Section 2.3.6.1.1.27
turret_move	Section 2.3.6.1.1.3
turret_get_ref_ind	Section 2.5.5.2.16
turret_set_stab_sys	Section 2.5.5.2.3
controls_turret_ref_ind	Section 2.3.2
alpha_send	Section 2.3.2.3.1

Table 2.3-247: turret_simul Information.

2.3.6.1.1.3 turret_move

This routine is called by **turret_simul**() to make the turret slew and the gun elevate. It checks to make sure that the subsystems are engaged and working before the routines which actually perform the slewing and elevating are called.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
sin_stab_azi_rot	REAL	sim_types.h
sin_stab_elev_rot	REAL	sim_types.h
elev_rate	REAL	sim_types.h
slew_rate	REAL	sim_types.h

Calls	
Function	**Where Described**
turret_get_stab_changes	Section 2.5.5.2.5
make_sound_of_no_slewing	Section 2.3.6.1.1.25
turret_calc_turret_slew	Section 2.3.6.1.1.7
turret_move_azimuth	Section 2.5.5.2.6
make_sound_of_no_elevating	Section 2.3.6.1.1.26
turret_calc_gun_elev	Section 2.3.6.1.1.9
turret_move_elevation	Section 2.5.5.2.7

Table 2.3-248: turret_move Information.

2.3.6.1.1.4 turret_get_turret_slew_rate

This routine returns the turret slew rate.

Return Values		
Return Value	**Type**	**Meaning**
turret_slew_rate/DELTA_T	REAL	The turret slew rate

Table 2.3-249: turret_get_turret_slew_rate Information.

2.3.6.1.1.5 turret_get_gun_elev_rate

This routine returns the gun elevation rate.

Return Values		
Return Value	**Type**	**Meaning**
gun_elev_rate/DELTA_T	REAL	The gun elevation rate

Table 2.3-250: turret_get_gun_elev_rate Information.

2.3.6.1.1.6 turret_handles_values

This routine is called by the handles module to pass on the values of the gun slew rates, the gun elevation rates, which handles are engaged, and whether the fast slew is on.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
gun_slew_rate	REAL	sim_types.h
gun_elevate_rate	REAL	sim_types.h
handles_engaged	int	Standard
fast_slew_on	int	Standard

Table 2.3-251: turret_handles_values Information.

2.3.6.1.1.7 turret_calc_turret_slew

This routine moves the turret in azimuth. In addition to slewing the turret, this routine is also responsible for checking to make sure that the turret does not move too fast. It also checks to see that sufficient hydraulic pressure is available before starting the turret move.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
control_handle	REAL	sim_types.h
sin_stab_azi_rot	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
total_slew_rate	register REAL	sim_types.h
elec_slew_percent	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
total_slew_rate	REAL	The total slew rate
Calls		
Function	**Where Described**	
calc_slew_from_handle	Section 2.3.6.1.1.8	
electsys_turret_traverse_request	Section 2.3.6.3.1.23	
sound_of_turret_traversing	Section 2.1.3.3.18	

Table 2.3-252: turret_calc_turret_slew Information.

2.3.6.1.1.8 calc_slew_from_handle

This routine is called by **turret_move_azimuth**() to determine how far to slew the turret, based on the deflection of the gunner or commander's handles. The parameter, *gun_slew_handle*, is the normalized handle displacement, where -1.0 is complete deflection to the right, +1 is complete deflection to the left, and 0.0 is centered.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
handle_disp	register REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
slew_rate	register REAL	sim_types.h
abs_slew_handle	register REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
slew_rate	REAL	The slew rate

Table 2.3-253: calc_slew_from_handle Information.

2.3.6.1.1.9 turret_calc_gun_elev

This routine moves the gun in elevation. In addition to elevating the gun, this routine is also responsible for checking that the gun is not moving too fast. The routine checks for sufficient hydraulic pressure before elevating the gun.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
control_handle	register REAL	sim_types.h
sin_stab_elev_rot	register REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
total_elev_rate	register REAL	sim_types.h
elev_percent	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
total_elev_rate	REAL	the total elevation rate
Calls		
Function	**Where Described**	
calc_elev_from_handle	Section 2.3.6.1.1.10	
eelctsys_turret_elevation_request	Section 2.3.6.3.1.22	
sound_of_gun_elevating	Section 2.1.3.3.17	

Table 2.3-254: turret_calc_gun_elev Information.

2.3.6.1.1.10 calc_elev_from_handle

This routine is called by **turret_move_elev()** to determine how far to elevate the gun, based on the deflection of the gunner or commander's handles. The parameter, *gun_elev_handle*, is the normalized handle displacement, where -1.0 is complete deflection to the right, +1 is complete deflection to the left, and 0.0 is centered.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
handle_disp	register REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
elev_rate	register REAL	sim_types.h
abs_elev_handle	register REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
elev_rate	REAL	The elevation rate

Table 2.3-255: calc_elev_from_handle Information.

2.3.6.1.1.11 turret_gyros_simul

This routine is called by **turret_simul**() to simulate the spinning up or spinning down of the turret gyros. The variable, *gyro_speed*, is a number between 0.0 and 1.0, representing the gyros' speed as a percentage of their full working speed.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
g_dir	pointer to register int	Standard
g_speed	pointer to register REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
turret_gyros_status	int	The status of the gyros
Calls		
Function	**Where Described**	
turret_gyros_status	Section 2.3.6.1.1.16	
controls_turret_drive_system_on	Section 2.3.2	

Table 2.3-256: turret_gyros_simul Information.

2.3.6.1.1.12 turret_stab_on

This routine called by the controls module when the stab switch is turned on.

Calls	
Function	**Where Described**
controls_turret_stab_on	Section 2.3.2

Table 2.3-257: turret_stab_on Information.

2.3.6.1.1.13 turret_stab_off

This routine called by the controls module when the stab switch is turned off.

Calls	
Function	**Where Described**
controls_turret_stab_off	Section 2.3.2

Table 2.3-258: turret_stab_off Information.

2.3.6.1.1.14 turret_gyros_spool_up

This routine called by the controls module when the turret power is turned on, in order to spool up the turret gyros.

2.3.6.1.1.15 turret_gyros_spool_down

This routine called by the controls module when the turret power is turned off, in order to spool fown the turret gyros.

Calls	
Function	**Where Described**
controls_turret_drive_system_off	Section 2.3.2

Table 2.3-259: turret_gyros_spool_down Information.

2.3.6.1.1.16 turret_gyros_status

This routine is called by the controls module to determine the state of the turret gyros.

Return Values		
Return Value	**Type**	**Meaning**
GYROS_SPOOLED_UP	int	The gyros are spooled up
GYROS_SPOOLED_DOWN	int	The gyros are spooled down
GYROS_STILL_SPOOLING	int	The gyros are in the process of spooling

Table 2.3-260: turret_gyros_status Information.

2.3.6.1.1.17 turret_break_elevation_drive

This routine causes the elevation drive to fail.

2.3.6.1.1.18 turret_repair_elevation_drive

This routine causes the elevation drive to be repaired.

2.3.6.1.1.19 turret_break_stab_system

This routine causes the stabilization system to fail.

2.3.6.1.1.20 turret_repair_stab_system

This routine causes the stabilization system to be repaired.

2.3.6.1.1.21 turret_break_mount_interface

This routine causes the mount interface to fail.

2.3.6.1.1.22 turret_repair_mount_interface

This routine causes the mount interface to be repaired.

2.3.6.1.1.23 turret_break_traverse_drive

This routine causes the traverse drive to fail.

2.3.6.1.1.24 turret_repair_traverse_drive

This routine causes the traverse drive to be repaired.

2.3.6.1.1.25 turret_collision_detected

This routine is called whenever kinematics whenever a collision is detected. It determines whether the gun was pointing in the direction of the collision. If so, the routine checks to see whether to break the turret-mount interface. When the turret-mount interface is broken, the gun cannot be elevated (except in emergency mode) or fired.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
agent_id	pointer to VehicleID	basic.h
event_id	long	Standard
coll_sector	register int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
azimuth	register REAL	sim_types.h
rel_sector	register int	Standard
t_to_h	register T_MAT_PTR	sim_types.h
Calls		
Function	**Where Described**	
rad_to_mil	Macro defined in sim_macros.h	
drivetrain_get_vehicle_speed	Section 2.3.6.2.4.8	
fail_break_system	Section 2.5.4.8.1	

Table 2.3-261: turret_collision_detected Information.

2.3.6.1.1.25 make_sound_of_no_slewing

This routine makes the sound of the turret not traversing.

Calls	
Function	**Where Described**
sound_of_turret_traversing	Section 2.1.3.3.18

Table 2.3-262: make_sound_of_no_slewing Information.

2.3.6.1.1.26 make_sound_of_no_elevating

This routine makes the sound of the gun not elevating.

Calls	
Function	**Where Described**
sound_of_gun_elevating	Section 2.1.3.3.17

Table 2.3-263: make_sound_of_no_elevating Information.

2.3.6.1.1.27 make_sound_of_no_turret_noise

This routine makes the sound of the gun not elevating and the turret not traversing.

Calls	
Function	**Where Described**
make_sound_of_no_slewing	Section 2.3.6.1.1.25
make_sound_of_no_elevatin g	Section 2.3.6.1.1.27

Table 2.3-264: make_sound_of_no_turret_noise Information.

2.3.6.1.1.28 turrret_get_gun_to_world

This routine returns the gun to world transformation matrix.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
g_to_w	T_MATRIX	sim_types.h
error	VECTOR	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
super_elevation	REAL	sim_types
Calls		
Function	**Where Described**	
bcs_get_super_elevation	Section 2.3.3.1	
turret_get_g_to_w	Section 2.5.5.2.13	

Table 2.3-265: turret_get_gun_to_world Information.

2.3.6.1.1.29 turret_tow_movement_off

This routine is called by the ammunition module when the 25mm gun is selected or no ammo is selected.

2.3.6.1.1.30 turret_tow_movement_on

This routine is called by the ammunition module when the TOW missile is selected.

2.3.6.1.1.31 turret_set_super_elevation

This routine is called by the ballistics computer system when the superelevation changes.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
new_super_elev	REAL	sim_types.h

Calls	
Function	**Where Described**
turret_move_elevation	Section 2.5.5.2.7

Table 2.3-266: turret_set_super_elevation Information.

2.3.6.1.2 m2_cupola.c

(./simnet/release/src/vehicle/m2/src/m2_cupola.c [m2_cupola.c])

The commander's periscope views in the M2 hatch are modelled by a rotating cupola with viewports attached to it. The commander turns the cupola by pressing a switch. The controls code determines the position of the cupola as a percentage of its full range. M2_cupola.c determine the angle of the cupola with respect to the turret from this information. The sine and cosine are made available to the CIG so the appropriate image can be drawn in the periscope viewports.

Includes:

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

Defines:

Symbol	Value	
CWS_FIELD_OF_VIEW	5.235987755	300 deg

REAL declarations and initialization:

cws_sin = 0.0
cws_cos = 1.0
cws_current_offset = 0.0

int declarations and initialization:

new_cws_value = TRUE

2.3.6.1.2.1 cupola_get_cws_cos_and_sin

This routine sets the values pointed to by *sine* and *cosine* to commander weapon system sine and cosine.

Parameters		
Parameter	Type	Where Typedef Declared
cosine	pointer to float	Standard
sine	pointer to float	Standard

Table 2.3-267: cupola_get_cws_cos_and_sin Information.

2.3.6.1.2.2 cupola_get_real_cws_cos_and_sin

This routine sets the values pointed to by *sine* and *cosine* to commander weapon system sine and cosine.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
cosine	pointer to REAL	sim_types.h
sine	pointer to REAL	sim_types.h

Table 2.3-268: cupola_get_real_cws_cos_and_sin Information.

2.3.6.1.2.3 convert_disp_to_angle

This routine sets the values pointed to by *psin* and *pcos* to the sine and cosine of the angle caluclated from the displacement and offset arguments (*disp* and *offset*)

Parameters		
Parameter	**Type**	**Where Typedef Declared**
disp	REAL	sim_types.h
fov	REAL	sim_types.h
psin	pointer to REAL	sim_types.h
pcos	pointer to REAL	sim_types.h
offset	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
angle	register REAL	sim_types.h

Table 2.3-269: convert_disp_to_angle Information.

2.3.6.1.2.4 cupola_cws_new_value

This routine sets the value of the commander weapon system offset to the value passed in *val*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h

Table 2.3-270: cupola_cws_new_value Information.

2.3.6.1.2.5 cupola_simul

This routine performs the tick by tick simulation of the cupola. The routine determines whether the cupola moved, and if so, the routine calculates the new commander weapon system sine and cosine.

Calls	
Function	**Where Described**
convert_disp_to_angle	Section 2.3.6.1.2.3

Table 2.3-271: cupola_simul Information.

2.3.6.1.2.6 cupola_init

This routine initializes the value of the commander weapon system offset at 0.0.

2.3.6.1.3 m2_ramp.c
(./simnet/release/src/vehicle/m2/src/m2_ramp.c [m2_ramp.c])

The level of fidelity of the M2 ramp model is rather low. The issues of concern are whether or not the ramp is moving and if it is fully open, fully closed or somewhere in between. It is not necessary to track the exact position. The model is maintained in m2_ramp.c. It responds to commands from controls to move or stop the ramp if possible. It tracks the time required to open or close the ramp and it makes the appropriate sounds associated with ramp movement and stopping. It tells the controls when a stop is reached and provides a routine to return the ramp status.

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libsound.h"
"m2_ramp.h"
"m2_cntrl.h"
"m2_sound_dfn.h"

Defines: DELTA_RAMP

Declarations:
ramp_val
ramp_going_up
ramp_going_down

2.3.6.1.3.1 ramp_init_ramp_down

This routine sets the *ramp_val* based on the position status of the ramp (either up or down).

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

Table 2.3-272: ramp_init_ramp_down Information.

2.3.6.1.3.2 ramp_simul

This routine is called on a tick by tick basis to provide the ramp simulation. The routine checks the position of the ramp. If the ramp is going up, the sound of the rear ramp stopping in the raised position is made; if the ramp is going down, the sound of the rear ramp stopping in the lowered position is made. If the ramp is fully raised, controls is notified to lock the ramp.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2
controls_ramp_unlocked_on	Section 2.3.2
controls_ramp_locked_on	Section 2.3.2

Table 2.3-273: ramp_simul Information.

2.3.6.1.3.3 ramp_up

This routine causes the ramp to be raised. The appropriate sound is made of the ramp being raised.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.3-274: ramp_up Information.

2.3.6.1.3.4 ramp_down

This routine causes the ramp to be lowered, making the appropriate sound.

Calls	
Function	**Where Described**
ssound_make_const_sound	Section 2.1.3.1.2

Table 2.3-275: ramp_down Information.

2.3.6.1.3.5 ramp_idle

This routine places the ramp in the idle state: moving neither up nor down. Any sounds of the ramp moving are ended.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.3-276: ramp_idle Information.

2.3.6.1.3.6 ramp_get_val

This routine returns the ramp status value.

Return Values		
Return Value	**Type**	**Meaning**
ramp_val	REAL	The ramp status

Table 2.3-277: ramp_get_va! Information.

2.3.6.1.3.7 ramp_down_status

This routine returns TRUE if the ramp is being lowered and FALSE if the ramp is not being lowered.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	The ramp is being lowered
FALSE	int	The ramp is not being lowered

Table 2.3-278: ramp_up_status Information.

2.3.6.1.4 m2_launcher.c
(./simnet/release/src/vehicle/m2/src/m2_launcher.c [m2_launcher.c])

M2_launcher.c maintains the model of the launcher. The level of fidelity required for the TOW launcher model is very close to that of the ramp. This file tracks whether the launcher is moving, in the up position, in the down position, or somewhere in between. It is not necessary to track the exact position. Thi file responds to commands from controls to move or stop the launcher, tracks the time required to raise or lower the launcher, and makes the appropriate sounds associated with launcher movement and stopping. The file tells controls when a stop is reached and provides a routine to return the launcher status. In addition, this file tells the network and the munitions management when the launcher is raised or lowered.

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libsound.h"
"libnetwork.h"
"m2_launcher.h"
"m2_cntrl.h"
"m2_sound_dfn.h"
"m2_ammo.h"
"m2_dtrain.h"

Defines:
DELTA_LAUNCHER
THREE_MILES_PER_HOUR

Declarations:
launcher_val
launcher_going_up
launcher_going_down

2.3.6.1.4.1 launcher_init_launcher_up

This routine sets the *launcher_val* based on the position status of the launcher (either up or down).

Parameters		
Parameter	Type	Where Typedef Declared
up_status	int	Standard

Table 2.3-279: launcher_init_launcher Information.

2.3.6.1.4.2 launcher_simul

This routine is called on a tick by tick basis to provide the launcher simulation. The launcher position during the last tick is checked. If the launcher is going up, the sound of the launcher stopping in the up position is made; if the launcher is going down, the sound of the launcher stopping in the down position is made. The network and ammo modules are notified of the launcher position.

Internal Variables		
Internal Variable	Type	Where Typedef Declared
temp	REAL	sim_types.h

Calls	
Function	Where Described
sound_make_const_sound	Section 2.1.3.1.2
controls_gunner_tow_launcher_off	Section 2.3.2
ammo_tow_launcher_off	Section 2.3.5.1.43
network_tow_launcher_down	Section 2.1.1.3.1.67.2
controls_gunner_tow_launcher_on	Section 2.3.2
ammo_tow_launcher_on	Section 2.3.5.1.42
network_tow_launcher_up	Section 2.1.1.3.1.67.1

Table 2.3-280: launcher_simul Information.

2.3.6.1.4.3 launcher_up

This routine causes the launcher to be raised if the vehicle speed is less than 3 mph. The appropriate sound is made of the launcher being raised.

Calls	
Function	Where Described
drivetrain_get_vehicle_speed	Section 2.3.6.2.4.8
sound_make_const_sound	Section 2.1.3.1.2

Table 2.3-281: launcher_up Information.

2.3.6.1.4.4 launcher_down

This routine causes the launcher to be lowered, making the appropriate sound.

Calls	
Function	Where Described
ssound_make_const_sound	Section 2.1.3.1.2

Table 2.3-282: launcher_down Information.

2.3.6.1.4.5 launcher_idle

This routine places the launcher in the idle state: moving neither up nor down. Any sounds of the launcher moving are ended.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.3-283: launcher_idle Information.

2.3.6.1.4.6 launcher_get_val

This routine returns the launcher status value.

Return Values		
Return Value	**Type**	**Meaning**
launcher_val	REAL	The launcher status

Table 2.3-284: launcher_get_val Information.

2.3.6.1.4.7 launcher_up_status

This routine returns TRUE if the laucher is being raised and FALSE if the launcher is not being raised.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	The launcher is going up
FALSE	int	The launcher is not going up

Table 2.3-285: launcher_up_status Information.

2.3.6.2 M2 Propulsion Simulation

The CSU's which provide the simulation routines are as follows:

m2_ptrain.c
m2_trans.c
m2_cntrllr.c
m2_dtrain.c
m2_engine.c
m2_engfail.c

2.3.6.2.1 m2_ptrain.c
(./simnet/release/src/vehicle/m2/src/m2_ptrain.c [m2_ptrain.c])

This module controls the overall simulation of the powertrain, ground dynamics and kinematics. The simulation routine "powertrain_simul()" is called from m2_main.c.

2.3.6.2.1.1 powertrain_init

This routine is called to initialize the elements of the M2 powertrain: the engine, the controller, the tranmission, the drivetrain, and the odometer.

Calls	
Function	**Where Described**
engine_init	Section 2.3.6.2.5.25
controller_init	Section 2.3.6.2.3.1
transmission_init	Section 2.3.6.2.2.3
drivetrain_init	Section 2.3.6.2.4.27
odometer_init	Section 2.3.2.3.4.1

Table 2.3-286: powertrain_init Information.

2.3.6.2.1.2 powertrain_simul

This routine is called on a tick by tick basis in order to provide the powertrain simulation. The engine, the controller, the transmission, the drivetrain, and the odometer module simulations are called from this routine.

Calls	
Function	**Where Described**
engine_simul	Section 2.3.6.2.5.1
controller_simul	Section 2.3.6.2.3.2
transmission_simul	Section 2.3.6.2.2.4
drivetrain_simul	Section 2.3.6.2.4.2
odometer_simul	Section 2.3.2.3.4.2

Table 2.3-287: powertrain_simul Information.

2.3.6.2.2 m2_trans.c

(./simnet/release/src/vehicle/m2/src/m2_trans.c [m2_trans.c])

The HMPT-500 Continuously Variable Ratio Transmission is modeled in these routine as two positive displacement fluid pumps coupled to the engine via two epicyclic gear trains. There are three forward ranges and one reverse range. Transmission oil leak, and major failure are simulated. The simulation routines are called from m2_ptrain.c.

Includes:

- "stdio.h"
- "sim_types.h"
- "sim_dfns.h"
- "sim_macros.h"
- "m2_cons.h"
- "m2_engine.h"
- m2_cntrllr.h"
- "m2_trans.h"
- "m2_dtrain.h"
- "m2_sound.h"
- "m2_main.h"
- "libkin.h"
- "libhull.h"
- "libsound.h"
- "libfail.h"
- "failure.h"

Declarations:

- pump
- trans
- Q_coeef
- T_coeff
- **fit_Q()**
- **fit_T()**
- **fit_Q_init()**
- **fit_T_init()**
- leak_timer
- mile_conter
- last_left_p
- last_right_p

Defines:

- **fit_R()**
- **fit_D()**
- TEN_MIN_OF_TICKS

2.3.6.2.2.1 transmission_break_transmission

If the transmission status is NORMAL, this routine causes the oil temperature to become high and the oil pressure to become low, resulting in a major transmission failure.

Calls	
Function	**Where Described**
controls_transmission_oil_temperature_high	Section 2.3.2
controls_oil_pressure_low	Section 2.3.2

Table 2.3-288: transmission_break_transmission Information.

2.3.6.2.2.2 transmission_replace_transmission

This routine repairs the transmission, causing the oil temperature and oil pressure to become normal.

Calls	
Function	**Where Described**
controls_transmission_oil_temperature_normal	Section 2.3.2
controls_transmission_oil_pressure_normal	Section 2.3.2

Table 2.3-289: transmission_replace_transmission Information.

2.3.6.2.2.3 transmission_init

This routine initializes the transmission module, including the load torque, the status, timers, counters, and the failures.

Calls	
Function	**Where Described**
fit_Q_init	Section 2.3.6.2.2.7
fit_T_init	Section 2.3.6.2.2.8
fail_init_failure	Section 2.5.4.11.2

Table 2.3-290: transmission_init Information.

2.3.6.2.2.4 transmission_simul

This routine is called on a tick by tick basis to simulate the transmission. The left and right stroke, the engine speed, the shaft speeds, the flow rate and fluid resistance, and the failuares are modeled. The transmission pressures and hydraulic loads are calculated. The load to engine is caluclated for the current gear (mechanical loads are proportional to hydraulic loads). The pressure limits are calculated after the loads to the engine. The torques applied to the shaft are calculated.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
left_stroke	REAL	sim_types.h
right_stroke	REAL	sim_types.h
a_left_stroke	REAL	sim_types.h
a_right_stroke	REAL	sim_types.h
e_speed	REAL	sim_types.h
left_shaft_speed	REAL	sim_types.h
right_shaft_speed	REAL	sim_types.h
Errors		
Error Name	**Reason for Error**	
Transmission_simul	Unknown gear setting	
Calls		
Function	**Where Described**	
controller_stroke_left	Section 2.3.6.2.3.6	
controller_stroke_right	Section 2.3.6.2.3.7	
engine_speed	Section 2.3.6.2.5.11	
drivetrain_left_omega	Section 2.3.6.2.4.9	
drivetrain_right_omega	Section 2.3.6.2.4.10	
controller_gear	Section 2.3.6.2.3.8	
fit_D	Section	
fit_R	Section	
fit_Q	Section 2.3.6.2.2.7	
fit_T	Section 2.3.6.2.2.8	
odometer_mile_counter	Section 2.3.2.3.4.8	
controls_transmission_oil_temperature_high	Section 2.3.2	
transmission_break_transmission	Section 2.3.6.2.2.1	

Table 2.3-291: transmission_simul Information.

2.3.6.2.2.5 fit_T_init

This routine initializes the coefficients for the torque fit.

2.3.6.2.2.6 fit_Q_init

This routine initializes the coefficients for the flow rate fit.

2.3.6.2.2.7 fit_Q

This routine models the flow rate fit using the flow rate coefficients and current gear, engine speed, and both shaft speeds. Parameters are represented as follows:

gear -- The current gear
e_speed -- The engine speed
shaft_speed -- This shaft speed
other_shaft_speed -- Other shaft speed

Parameters		
Parameter	**Type**	**Where Typedef Declared**
gear	int	Standard
e_speed	REAL	sim_types.h
shaft_speed	REAL	sim_types.h
other_shaft_speed	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
coeff_P	pointer to REAL	sim_types.h
Q	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
Q	REAL	The flow rate fit

Table 2.3-292: fit_Q Information.

2.3.6.2.2.8 fit_T

This routine models the torque output fit using the torque coefficients and current gear, and the pressure in the two pump motors. Parameters are represented as follows:

gear -- The current gear
this_P -- The pressure in this pump motor
other_P -- The pressure in the other pump motor

Parameters		
Parameter	**Type**	**Where Typedef Declared**
gear	int	Standard
this_P	REAL	sim_types.h
other_P	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
T	REAL	sim_types.h
coeef_P	pointer to REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
T	REAL	The torque output fit

Table 2.3-293: fit_T Information.

2.3.6.2.2.9 transmission_load_torque

This routine returns the transmission load torque.

Return Values		
Return Value	**Type**	**Meaning**
trans.load_torque	REAL	The transmission load torque

Table 2.3-294: transmission_load_torque Information.

2.3.6.2.2.10 transmission_torque_left

This routine returns the the torque applied to the left side of the transmission.

Return Values		
Return Value	**Type**	**Meaning**
trans.left.T	REAL	The left side of the transmission's torque value

Table 2.3-295: transmission_torque_left Information.

2.3.6.2.2.11 transmission_torque_right

This routine returns the torque applied to the right side of the transmission.

Return Values		
Return Value	**Type**	**Meaning**
trans.load_torque	REAL	The right side of the transmission's torque value

Table 2.3-296: transmission_load_torque Information.

2.3.6.2.2.12 transmission_oil_leak

This routine models an oil leak in the transmission, setting the leak status to ISFAILED. The controls of transmission oil pressure are set to low and the odometer is reset.

Calls	
Function	**Where Described**
controls_transmission_oil_pressure_low	Section 2.3.2
odometer_mile_counter_reset	Section 2.3.2.3.4.7

Table 2.3-297: transmission_oil_leak Information.

2.3.6.2.2.13 transmission_repair_oil_leak

This routine causes the repair of a transmission oil leak, setting the leak status to NORMAL. The transmission oil temperature and pressure controls are set to normal.

Calls	
Function	**Where Described**
controls_transmission_oil_temperature_normal	Section 2.3.2
controls_transmission_oil_pressure_normal	Section 2.3.2

Table 2.3-298: transmission_repair_oil_leak Information.

2.3.6.2.2.14 Debugging tools

The following routines are used for printing tranmission status reports during debugging:

tranmission_dump()
tranmission_banner()
tranmission_data_title()
tranmission_data_banner()
tranmission_data_dump()

2.3.6.2.3 m2_cntrllr.c

(/simnet/release/src/vehicle/m2/src/m2_cntrllr.c [m2_cntrllr.c])

Includes:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"m2_engine.h"
"m2_cntrllr.h"
"m2_trans.h"
"m2_cons.h"
"m2_launcher.h"

In the following header structure, cntrllr, *gear* is the gear the transmission is in, *gear_set* is the gear the transmission is set to, *upshift_delay* is the number of ticks to delay upshifting from 1st to 2nd after a downshift, *s_cms* is the steering-commanded input, *cam* is the cam position, *throttle* is the throttle value to the engine, *speed_ref* is the reference speed in rad/s, and *speed_error* is *speed_ref* minus *actual_speed* in rad/s.

Item	Type	Where Type Defined
gear	int	Standard
gear_set	int	Standard
upshift_delay	int	Standard
s_cmd	REAL	sim_types.h
cam	REAL	sim_types.h
throttle	REAL	sim_types.h
speed_ref	REAL	sim_types.h
speed_error	REAL	sim_types.h

Table 2.3-299: cntrllr Structure Definition

In the following header structure, stroke, *left* is the left stroke value, *right* is the right stroke value, and *val* is the stroke value from the control cam.

Item	Type	Where Type Defined
left	REAL	sim_types.h
right	REAL	sim_types.h
val	REAL	sim_types.h

Table 2.3-300: stroke Structure Definition

In the following header structure, steer, *val* is the steering val = bar**2, *gov_input* is the steering governor input, and *mod* is the steering modifier term.

Item	Type	Where Type Defined
val	REAL	sim_types.h
gov_input	REAL	sim_types.h
mod	REAL	sim_types.h

Table 2.3-301: steer Structure Definition

In the following header structure, cam_cons[5], *min_cam* is the minimum cam position for gear setting, *max_cam* is the maximum cam position for gear setting, *shift_up* is the up shift cam position, *shift_dn* is the down shift cam position, *mins* is the minimum stroke, *maxs* is the maximum stroke, and *Kg* is the error gain for this gear.

Item	Type	Where Type Defined
min_cam	REAL	sim_types.h
max_cam	REAL	sim_types.h
shift_up	REAL	sim_types.h
shift_dn	REAL	sim_types.h
mins	REAL	sim_types.h
maxs	REAL	sim_types.h
Kg	REAL	sim_types.h

Table 2.3-302: cam_cons[5] Structure Definition

Defines:

steer_gov(x) ((REAL) 0.722 *(((REAL) 1.482 - (x))/((REAL) 1.07 + (x))))

Procedure declarations:

shift_check()
stroke_calc()
fit_speed_ref()

2.3.6.2.3.1 controller_init

This routine initializes the transmission controller.

2.3.6.2.3.2 controller_simul

This routine provides the tick by tick simulation of the transmission controller.

Internal Variables		
Internal Variable	Type	Where Typedef Declared
temp	REAL	sim_types.h
Calls		
Function	Where Described	
fit_speed_ref	Section 2.3.6.2.3.3	
engine_speed	Section 2.3.6.2.5.11	
max	sim_macros.h	
min	sim_macros.h	
launcher_up_status	Section 2.3.6.1.4.7	
shift_check	Section 2.3.6.2.3.4	
stroke_calc	Section 2.3.6.2.3.5	

Table 2.3-303: controller_simul Information.

2.3.6.2.3.3 fit_speeed_ref

This routine calculates the reference speed from the throttle value and the cam value.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
throttle	REAL	sim_types.h
main_cam	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
speed_ref	REAL	sim_types.h
max_throttle	REAL	sim_types.h
min_throttle	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
M2_IDLE_REF + speed_ref * M2_REF_SPEED_SLOPE	REAL	The reference speed
Calls		
Function	**Where Described**	
min	sim_macros.h	
max	sim_macros.h	

Table 2.3-304: fit_speeed_ref Information.

2.3.6.2.3.4 shift_check

This routine shifts the gears if a shift is required, making sure the vehicle is in the correct gear. Drive is the only gear that shifting is allowed.

2.3.6.2.3.5 stroke_calc

This routine calculates the stroke and sets the appropriate stroke value holders.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp	REAL	sim_types.h

Errors	
Error Name	**Reason for Error**
stroke_calc, Unknown gear ...	The element cntrllr.gear has an unexpected value.

Calls	
Function	**Where Described**
error_printf	Section 2.6.4.14.1
min	sim_macros.h
max	sim_macros.h
abs	sim_macros.h
steer_gov	Section 2.3.2

Table 2.3-305: stroke_calc Information.

2.3.6.2.3.6 controller_stroke_left

This routine returns the value of the left stroke.

Return Values		
Return Value	**Type**	**Meaning**
stroke.left	REAL	The value of the left stroke.

Table 2.3-306: controller_stroke_left Information.

2.3.6.2.3.7 controller_stroke_right

This routine returns the value of the right stroke.

Return Values		
Return Value	**Type**	**Meaning**
stroke.right	REAL	The value of the right stroke

Table 2.3-307: controller_stroke_right Information.

2.3.6.2.3.8 controller_gear

This routine returns the transmission controller gear value.

Return Values		
Return Value	**Type**	**Meaning**
cntrllr.gear	int	The controller gear value

Table 2.3-308: controller_gear Information.

2.3.6.2.3.9 controller_neutral

This routine sets the transmission gear value to neutral and disengages the start of the engine.

Calls	
Function	**Where Described**
engine_out_of_start	Section 2.3.6.2.5.3

Table 2.3-309: controller_neutral Information.

2.3.6.2.3.10 controller_pivot

The routine sets the transmission gear value to neutral.

2.3.6.2.3.11 controller_drive

This routine places the gear setting in drive position.

2.3.6.2.3.12 controller_low

This routine places the transmission gear setting in low speed.

2.3.6.2.3.13 controller_reverse

This routine places the transmission gear setting in reverse.

2.3.6.2.3.14 controller_start

This routine sets the transmission gear value to neutral and starts the engine.

Calls	
Function	**Where Described**
engine_start	Section 2.3.6.2.5.2

Table 2.3-310: controller_start Information.

2.3.6.2.3.15 controller_set_throttle

This routine sets the throttle to the value passed in *val*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h

Table 2.3-311: controller_set_throttle Information.

2.3.6.2.3.16 controller_set_steering_bar

This routine sets the transmission steering to the value passed in *val*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h
Calls		
Function	**Where Described**	
abs	sim_macros.h	

Table 2.3-312: controller_set_steering_bar Information.

2.3.6.2.3.17 cntrllr_dump

This routine prints the values of stroke, left stroke, right stroke, cam, and gear.

2.3.6.2.3.18 cntrllr_banner

This routine prints the banner heading for the controller dump.

2.3.6.2.3.19 cntrllr_data_title

This routine prints the data title.

2.3.6.2.3.20 cntrllr_data_banner

This routine prints the data banner.

2.3.6.2.3.21 cntrllr_data_dump

This routine prints out the values of the left stroke and the right stroke.

2.3.6.2.4 m2_dtrain.c

(./simnet/release/src/vehicle/m2/src/m2_dtrain.c [m2_dtrain.c])

Track and hull dynamics are simulated in this file. The transmission puts out torque to the tracks which accelerate and rotate the tank. Brakes are also accounted for in this file. Failure of service brake, parking brake, and tracks are maintained. These simulation routines are called from m2_ptrain.c.

This file contains variables, functions, arrays, and structures. The variables are defined and initialized as shown in the next table.

Variable	Type	Where Type Defined	Initial Value
M2_TRACK_ FAILURE_DRAG	REAL	sim_types.h	100000.0
coefficient_of_ traction[2][6]	REAL	sim_types.h	N/A
soil_type	int	Standard	N/A
track_speed_fraction	REAL	sim_types.h	N/A

Table 2.3-313: m2_dtrain.c Variable Definitions.

The dtrain_status structure is described below. It gives the status of the drivetrain components.

Item	Type	Where Type Defined
slip_state	int	Standard
left_track	int	Standard
right_track	int	Standard
brake	int	Standard
p_brake	int	Standard
parking_brake_on	int	Standard

Table 2.3-314: dtrain_status Structure Definition.

The dtrain structure is described below.

Item	Type	Where Type Defined
udot[4]	REAL	sim_types.h
u[4]	REAL	sim_types.h
FO	REAL	sim_types.h
friction_FO	REAL	sim_types.h
T1	REAL	sim_types.h
friction_T1	REAL	sim_types.h
T2	REAL	sim_types.h
friction_T2	REAL	sim_types.h
T3	REAL	sim_types.h
friction_T3	REAL	sim_types.h
forward_diff	REAL	sim_types.h
pitch_sin	REAL	sim_types.h
cant_sin	REAL	sim_types.h
traction_force	REAL	sim_types.h
brake_factor	REAL	sim_types.h

Table 2.3-315: dtrain Structure Definition.

The track_friction[6] and rotational_friction[6] static structures are described below.

Item	Type	Where Type Defined
Staticf	REAL	sim_types.h
Viscous	REAL	sim_types.h

Table 2.3-316: track_friction[6] and rotational_friction[6] Structure Definition.

Slip flags are defined as follows:

Constant	Value
NO_SLIP	0
SLIP	1

Table 2.3-317: m2_dtrain.c Constant Definitions.

The arrays are defined and initialized as shown in the next table.

Array	Type	Where Type Defined	Initial Values
rear_wheel	VECTOR	sim_types.h	{0.0, -4.5, 0.0}
left_wheel	VECTOR	sim_types.h	{-1.75, 4.5, 0.0}
right_wheel	VECTOR	sim_types.h	{1.75, 4.5, 0.0}

Table 2.3-318: m2_dtrain.c Array Definitions.

These parameters are used to initialize the suspension library at run-time.

Parameter	Value

ROT_WN -- rot suspension natural freq (rad)	(REAL)2.0
ROT_ZETA -- rotational suspension damping ratio	(REAL)0.3
SIDE_WN -- side suspension natural freq (rad)	(REAL)3.5
SIDE_ZETA -- side suspension damping ratio	(REAL)0.2
LEVER_ARM -- Meters	(REAL)0.003
ANGLE_LIM -- ~9 degrees .7 m by 4.5 m	(REAL)0.156
GUN_FORCE -- Force of firing the gun	(REAL)0.01
M2_NO_SLIP_LUMPED_MASS	(M2_MASS+2.0*M2_TRACK_MASS+(M2_WHEEL_INERTIA_0*2.0)/(M2_WHEEL_RADIUS*M2_WHEEL_RADIUS))
M2_NO_SLIP_LUMPED_INERTIA	(M2_INERTIA_2+2.0*M2_WHEEL_INERTIA_2+M2_TRACK_MASS*M2_WIDTH*M2_WIDTH*0.5+(M2_WIDTH*M2_WIDTH*M2_WHEEL_INERTIA_0)/(2.0*M2_WHEEL_RADIUS*M2_WHEEL_RADIUS))
M2_SLIP_LUMPED_VEHICLE_MASS	(M2_MASS+2.0*M2_TRACK_MASS)
MAX_TRACK_CANT_SIN -- SIN(22 DEGREES) = 40%	0.374606
TRACK_SELF_REPAIR_TIME -- minutes	30

Table 2.3-319: m2_dtrain.c Run-time Initialization Parameters.

2.3.6.2.4.1 check_for_thrown_track

This routine checks for thrown tracks. Each time one is found, the fail_break_system routine is called.

Calls	
Function	**Where Described**
fail_break_system	Section 2.5.4.8.1

Table 2.3-320: check_for_thrown_track Information.

2.3.6.2.4.2 drivetrain_simul

This routine simulates vehicle drive train functions.

Calls	
Function	**Where Described**
transmission_torque_left	Section 2.3.6.2.2.10
transmission_torque_right	Section 2.3.6.2.2.11
terrain_get_terrain_type	Section 2.5.11.3.1
kinematics_pitch_sin	Section 2.5.8.5.1
kinematics_cant_sin	Section 2.5.8.5.3
check_for_thrown_track	Section 2.3.6.2.4.1
rotational_friction_factor	Section 2.3.6.2.4.3
compute_traction_force	Section 2.3.6.2.4.4
check_forward_collision	Section 2.3.6.2.4.6
check_side_collision	Section 2.3.6.2.4.7
check_for_slip	Section 2.3.6.2.4.5
suspension_acceleration_is	Section 2.5.6.3.1
sound_of_tracks	Section 2.1.3.3.13
meter_speed_set	Section 2.3.2.3.3
kinematics_move_vehicle	Section 2.5.8.7.1
kinematics_turn_vehicle	Section 2.5.8.11.1
network_set_dust_cloud	Section 2.1.1.3.1.12.1
tracks_get_dust_cloud	Section 2.3.7.2.1

Table 2.3-321: drivetrain_simul.

2.3.6.2.4.3 rotational_friction_factor

This routine calculates the rotational friction factor from vehicle speed (v) and angular velocety (w).

Parameters		
Parameters	**Type**	**Where Typedef Declared**
v	REAL	sim_types.h
w	REAL	sim_types.h
ReturnValues		
Return Value	**Type**	**Meaning**
factor	REAL	Rotational friction factor.

Table 2.3-322: rotational_friction_factor.

2.3.6.2.4.4 compute_traction_force

This routine calculates the traction force from the vehicle pitch, vehicle state, and soil type.

Parameters		
Parameters	**Type**	**Where Typedef Declared**
pitch_sin	REAL	sim_types.h
state	int	Standard
soil_type	int	Standard
ReturnValues		
Return Value	**Type**	**Meaning**
traction_force	REAL	Traction force.

Table 2.3-323: compute_traction_force.

2.3.6.2.4.5 check_for_slip

This routine determines whether the vehicle drive tracks are slipping or not given the traction force and soil type.

Parameters		
Parameters	**Type**	**Where Typedef Declared**
traction_force	REAL	sim_types.h
ReturnValues		
Return Value	**Type**	**Meaning**
SLIP	int	Drive tracks are slipping.
NO_SLIP	int	Drive tracks are not slipping.

Table 2.3-324: check_for_slip.

2.3.6.2.4.6 check_forward_collision

This routine determines whether a collision has occurred while traveling backwards or forwards, given the velocety *v*.

Parameters		
Parameters	**Type**	**Where Typedef Declared**
v	REAL	sim_types.h
ReturnValues		
Return Value	**Type**	**Meaning**
0.0	REAL	Collision has occurred.
v	REAL	No collision has occurred.
Calls		
Function	**Where Described**	
collision_rear_collision	Section 2.5.10.4.3	
collision_left_collision	Section 2.5.10.4.1	
collision_right_collision	Section 2.5.10.4.2	

Table 2.3-325: check_forward_collision.

2.3.6.2.4.7 check_side_collision

This routine determines whether a side collision has occurred while the vehicle is turning.

Parameters		
Parameters	**Type**	**Where Typedef Declared**
w	REAL	sim_types.h
ReturnValues		
Return Value	**Type**	**Meaning**
0.0	REAL	Collision has occurred.
w	REAL	No collision has occurred.
Calls		
Function	**Where Described**	
collision_left_collision	Section 2.5.10.1.1	
collision_right_collision	Section 2.5.10.1.2	

Table 2.3-326: check_side_collision.

2.3.6.2.4.8 drivetrain_get_vehicle_speed

This routine returns the vehicle speed.

ReturnValues		
Return Value	Type	Meaning
dtrain.u[0]	REAL	Vehicle speed.

Table 2.3-327: drivetrain_get_vehicle_speed.

2.3.6.2.4.9 drivetrain_left_omega

This routine returns the angular velocety of the left track.

ReturnValues		
Return Value	Type	Meaning
dtrain.u[2]	REAL	angular velocety of left track

Table 2.3-328: drivetrain_left_omega.

2.3.6.2.4.10 drivetrain_right_omega

This routine returns the angular velocety of the right track.

ReturnValues		
Return Value	Type	Meaning
dtrain.u[3]	REAL	angular velocety of right track

Table 2.3-329: drivetrain_right_omega.

2.3.6.2.4.11 drivetrain_set_brake

This routine sets the brake factor to *val* if the parking brake is off and the service brake has not failed. *val* is a control setting from the brake pedal.

Parameters		
Parameters	Type	Where Typedef Declared
val	REAL	sim_types.h

Table 2.3-330: drivetrain_set_brake.

2.3.6.2.4.12 drivetrain_parking_brake_set

This routine sets the parking brake on and makes the sound of the brake being set.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.3-331: drivetrain_parking_brake_set.

2.3.6.2.4.13 drivetrain_parking_brake_release

This routine sets the parking brake off and the brake factor to 0 and makes the sound of the parking brake being released.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.3-332: drivetrain_parking_brake_release.

2.3.6.2.4.14 drivetrain_service_brake_failure

This routine causes the service brake to fail.

2.3.6.2.4.15 drivetrain_repair_service_brake

This routine repairs the service brake.

2.3.6.2.4.16 drivetrain_parking_brake_failure

This routine fails the parking brake and sets the brake factor to 0.

Calls	
Function	**Where Described**
sound_make_const_sound	Section 2.1.3.1.2

Table 2.3-333: drivetrain_parking_brake_failure.

2.3.6.2.4.17 drivetrain_repair_parking_brake

This routine repairs the parking brake and sets it on with a brake factor of 1.

2.3.6.2.4.18 drivetrain_throw_right_track

This routine fails the right drivetrain track.

Calls	
Function	**Where Described**
bigwheel_right_track_broken	Section 2.5.10.10.2

Table 2.3-334: drivetrain_throw_right_track.

2.3.6.2.4.19 drivetrain_throw_left_track

This routine fails the left drivetrain track.

Calls	
Function	**Where Described**
bigwheel_left_track_broken	Section 2.5.10.10.1

Table 2.3-335: drivetrain_throw_left_track.

2.3.6.2.4.20 drivetrain_repair_thrown_tracks

This routine repairs both the left and right drivetrain tracks, regardless of which, if any, are broken.

Calls	
Function	**Where Described**
bigwheel_repair_tracks	Section 2.5.10.10.3

Table 2.3-336: drivetrain_repair_thrown_tracks.

2.3.6.2.4.21 dump_drivetrain_state

This routine prints the status of the drivetrain brakes and tracks.

2.3.6.2.4.22 drivetrain_data_title

This routine prints the "dtrain" header for debugging.

2.3.6.2.4.23 drivetrain_data_banner

This routine prints the column headings Wl, Wr, and V for debugging.

2.3.6.2.4.24 drivetrain_data_dump

This routine prints the drivetrain data Wl, Wr, and V for debugging.

2.3.6.2.4.25 drivetrain_banner

This routine prints the column headings Tl, Tr, Wl, Wr, V, and Rot for debugging.

2.3.6.2.4.26 drivetrain_dump

This routine prints the drivetrain data Tl, Tr, Wl, Wr, V, and Rot for debugging.

Calls	
Function	**Where Described**
transmission_torque_left	Section 2.3.6.2.2.10
transmission_torque_right	Section 2.3.6.2.2.11

Table 2.3-337: drivetrain_dump.

2.3.6.2.4.27 drivetrain_init

This routine initializes the drivetrain.

Calls	
Function	**Where Described**
bigwheel_veh_init	Section 2.5.10
suspension_params	Section 2.5.6.5.1
fail_init_failure	Section 2.5.4.11.2

Table 2.3-338: drivetrain_init Information

2.3.6.2.5 m2_engine.c

(./simnet/release/src/vehicle/m2/src/m2_engine.c [m2_engine.c])

The Cummins VTA-903-T diesel engine is simulated in this module by parameterizing a family of fourth order polynomial torque curves based on throttle position. The engine runs its own dynamics based on load torque from the transmission, output torque, and engine inertia. The simulation routines are called from m2_ptrain.c.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"m2_engfail.h"
"libfail.h"
"failure.h"
"m2_trans.h"
"m2_cons.h"
"m2_cntrllr.h"
"m2_sound.h"
"m2_fuelsys.h"
"m2_gages.h"
"libsound.h"

In the following header structure, engine, *state* is the engine state (OFF, CRANK, RUN), *start_delay* is the number of ticks to delay starting, *fuel_supply* is the fuel available flag, *start_state* is the engine starting states, *pressure_status* is the oil pressure (normal or low), *coolant_status* is the coolant temperature (normal or high), *accessories* is the engine accessory switch, *oil_pressure* is the oil pressure (0 to 100 psi), *oil_temperature* and *coolant_temperature* are in Farenheit, *speed* is in rad/sec, *accel* is in rad/sec**2, *torque* is in kg-m**2/sec**2, *power_percent* is the power percentage, *load_torque* is in Newton-meters, *throttle* is throttle from controls, *fuel_flow* is in gallons/hour, *inertia[5]* is the inertia term for each gear, *engine_failure* is the engine failure flag, and *starter_failure* is the starter failure flag.

Item	Type	Where Type Defined
state	int	Standard
start_delay	int	Standard
fuel_supply	int	Standard
start_state	int	Standard
pressure_status	int	Standard
coolant_status	int	Standard
accessories	int	Standard
oil_pressure	REAL	sim_types.h
oil_temperature	REAL	sim_types.h
coolant_temperature	REAL	sim_types.h
speed	REAL	sim_types.h
accel	REAL	sim_types.h
torque	REAL	sim_types.h
power_percent	REAL	sim_types.h
load_torque	REAL	sim_types.h
throttle	REAL	sim_types.h
fuel_flow	REAL	sim_types.h
inertia[5]	REAL	sim_types.h
engine_failure	int	Standard
starter_failure	int	Standard

Table 2.3-339: engine Structure Definition

Defines:

Symbol	Value
ENGINE_OFF	(int) -1
ENGINE_CRANK	(int) 0
ENGINE_RUN	(int) 1
STARTER_NO_CRANK	(int) -1
STARTER_CRANK	(int) 0
STARTER_START	(int) 1
HIGH_COOLANT_TEMP	220.0
LOW_OIL_PRESSURE	4.5
MAX_OVERHEAT_TICKS	27000

Procedure declarations:

fit_engine_torque()
engine_run()
engine_crank()
engine_off()

int declarations:
overheat_ticks

REAL declarations:
engine_speed_fraction

2.3.6.2.5.1 engine_simul

This routine provides the engine simulation on a tick by tick basis. The engine performance is simulated in the different engine states. The failure timers are updated, and the fuel supply is regulated.

Errors	
Error Name	**Reason for Error**
engine_simul, Unknown state ...	The element engine.state has an unexpected value.
Calls	
Function	**Where Described**
controls_engine_oil_pressure_low	Section 2.3.2
engine_crank	Section 2.3.6.2.5.17
engine_run	Section 2.3.6.2.5.16
max	sim_macros.h
gage_oil_temperature	Section 2.3.2.3.2.2
gage_coolant_temperature	Section 2.3.2.3.2.3
controls_coolant_temperature_high	Section 2.3.2
controls_coolant_temperature_normal	Section 2.3.2
engine_coolant_leak	Section 2.3.6.2.6.1
error_printf	Section 2.6.4.14.1
gage_oil_pressure	Section 2.3.2.3.2.1
meter_press_set	Section 2.3.2.3.3.6
meter_temp_set	Section 2.3.2.3.3.5
controls_engine_oil_pressure_normal	Section 2.3.2
engine_failure_update	Section 2.3.6.2.6.11
fuel_set_flow	Section 2.3.5.2.5

Table 2.3-340: controller_simul Information.

2.3.6.2.5.2 engine_start

This routine starts the engine from the ENGINE_OFF state, and performs no operation from any other state.

Errors	
Error Name	**Reason for Error**
engine_start, Unknown state ...	The element engine.state has an unexpected value.
Calls	
Function	**Where Described**
error_printf	Section 2.6.4.14.1

Table 2.3-341: engine_start Information.

2.3.6.2.5.3 engine_out_of_start

This routine is called to disengage the start.

Calls	
Function	**Where Described**
sound_of_engine_cranking_stop	Section 2.1.3.3.11
sound_of_engine_stall	Section 2.1.3.3.12

Table 2.3-342: engine_out_of_start Information.

2.3.6.2.5.4 engine_accessory_on

This routine sets the engine accessories and engine accessory controls.

Calls	
Function	**Where Described**
controls_engine_accessory_on	Section 2.3.2

Table 2.3-343: engine_accessory_on Information.

2.3.6.2.5.5 engine_accessory_off

This routine turns off engine accessories and unsets the engine accessory controls.

Calls	
Function	**Where Described**
controls_engine_accessory_off	Section 2.3.2

Table 2.3-344: engine_accessory_off Information.

2.3.6.2.5.6 engine_fail

This routine causes the engine to fail.

2.3.6.2.5.7 engine_fix

This routine repairs the engine.

2.3.6.2.5.8 engine_starter_fail

This routine causes the engine starter to fail.

2.3.6.2.5.9 engine_starter_fix

This routine repairs the engine starter.

2.3.6.2.5.10 engine_running

This routine returns engine state: either ON or OFF.

Return Values		
Return Value	**Type**	**Meaning**
ON	int	Engine is running.
OFF	int	Engine is not running.

Table 2.3-345: engine_running Information.

2.3.6.2.5.11 engine_speed

This routine returns the engine speed.

Return Values		
Return Value	**Type**	**Meaning**
engine.speed	REAL	Engine speed.

Table 2.3-346: engine_speed Information.

2.3.6.2.5.12 engine_get_speed

This routine returns the engine speed in rpm.

Return Values		
Return Value	**Type**	**Meaning**
RADSEC_TO_RPM * engine.speed	REAL	Engine speed in rpm.

Table 2.3-347: engine_get_speed Information.

2.3.6.2.5.13 engine_rpm

This routine returns the engine speed in rpm.

Return Values		
Return Value	**Type**	**Meaning**
RADSEC_TO_RPM * engine.speed	REAL	Engine speed in rpm.

Table 2.3-348: engine_rpm Information.

2.3.6.2.5.14 engine_get_max_power

This routine returns the engine's maximum power.

Return Values		
Return Value	**Type**	**Meaning**
engine.power_percent	REAL	Engine power percentage.

Table 2.3-349: engine_get_max_power Information.

2.3.6.2.5.15 engine_set_throttle

This routine sets the engine's throttle equal to the input *val*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h

Table 2.3-350: engine_set_throttle Information.

2.3.6.2.5.16 engine_run

This routine sets the engine's torque, load, acceleration, and speed. The routine calls for sound generation of for the particular speed.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
throttle	REAL	sim_types.h

Calls	
Function	**Where Described**
fit_engine_torque	Section 2.3.6.2.5.19
transmission_load_torque	Section 2.3.6.2.2.9
controller_gear	Section 2.3.6.2.3.8
engine_off	Section 2.3.6.2.5.18
sound_of_engine	Section 2.1.3.3.16

Table 2.3-351: engine_run Information.

2.3.6.2.5.17 engine_crank

This routine is called to start the engine. Routines are called to generate cranking sound effects according to the starting state of the engine.

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

Calls	
Function	**Where Described**
electsys_engine_start_request	Section 2.2.6.3.1.8
sound_of_engine_cranking_start	Section 2.1.3.3.10
sound_of_engine_cranking_stop	Section 2.1.3.3.11
sound_of_engine_start	Section 2.1.3.3.14

Table 2.3-352: engine_crank Information.

2.3.6.2.5.18 engine_off

This routine sets the engine speed to zero, sets the engine state to off, calls a routine to generate the engine stop sound.

Calls	
Function	**Where Described**
sound_of_engine_stop	Section 2.1.3.3.15

Table 2.3-353: engine_off Information.

2.3.6.2.5.19 fit_engine_torque

This routine calculates and returns torque using the values of throttle and speed passed as the arguments.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
throttle	REAL	sim_types.h
speed	REAL	sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
x	REAL	sim_types.h
a	array of 4 REAL	sim_types.h
torque	REAL	sim_types.h
torque_gov	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
M2_ENG_NO_FUEL_TORQ	REAL	Engine fuel supply is off.
torque	REAL	The engine torque
Calls		
Function	**Where Described**	
max	sim_macros.h	
min	sim_macros.h	
engine_power_loss	Section 2.3.6.2.6.13	

Table 2.3-354: fit_engine_torque Information.

2.3.6.2.5.20 engine_dump

This routine prints engine speed, acceleration, torque, power, and load.

Calls	
Function	**Where Described**
transmission_load_torque	Section

Table 2.3-355: engine_dump Information.

2.3.6.2.5.21 engine_banner

This routine prints column headings for engine speed, acceleration, torque, power, and load.

2.3.6.2.5.22 engine_data_title

This routirie prints the title, Engine.

2.3.6.2.5.23 engine_banner

This routine prints column headings for speed and fuel.

2.3.6.2.5.24 engine_data_dump

This routine prin's engine speed and fuel flow.

2.3.6.2.5.25 engine_init

This routine initializes the engine structure variables.

Calls	
Function	**Where Described**
fail_init_failure	Section 2.5.4.11.2

Table 2.3-356: engine_init Information.

2.3.6.2.6 m2_engfail.c
(./simnet/release/src/vehicle/m2/src/m2_engfail.c [m2_engfail.c)

Failures are simulated in m2_engfail.c, and consist of: air filter, oil, coolant, fuel filter, and starter. The simulation routines are called from m2_ptrain.c.

Includes:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"m2_cons.h"
"m2_engine.h"

Defines:

Failure areas	Value
AIR_FILTER	(int) 0
FUEL_FILTER	(int) 1
COOLANT	(int) 2
OIL	(int) 3
STARTER	(int) 4
NUM_ENG_FAILURES	(int) 5
FAIL_STAT	struct fail_stat

In the following header structure, tagged fail_stat, "status" is NORMAL or ISFAILED, and "timer" is the number of ticks since the failure occurred.

Item	Type	Where Type Defined
status	int	Standard
timer	int	Standard

Table 2.3-357: failures[NUM_ENG_FAILURES] Structure Definition

Procedure declarations:
check_starter_failure()
check_engine_failure()

REAL declarations:
initial_power_percent

2.3.6.2.6.1 engine_coolant_leak

This routine sets control indicators and status to low level of coolant. The engine starter and engine are failed.

Calls	
Function	**Where Described**
controls_coolant_level_low	Section 2.3.2
engine_starter_fail	Section 2.3.6.2.5.8
engine_fail	Section 2.3.6.2.5.6

Table 2.3-358: engine_coolant_leak Information.

2.3.6.2.6.2 engine_coolant_normal

This routine returns the control indicators of coolant engine status to the normal level. The engine starter and engine are repaired.

Calls	
Function	**Where Described**
controls_coolant_level_normal	Section 2.3.2
engine_starter_fix	Section 2.3.6.2.5.9
engine_fix	Section 2.3.6.2.5.7

Table 2.3-359: engine_coolant_normal Information.

2.3.6.2.6.3 engine_clog_fuel_filter

This routine sets control indicators and status to fuel filter clogged. The engine starter and engine are failed.

Calls	
Function	**Where Described**
controls_fuel_filter_clogged	Section 2.3.2
engine_starter_fail	Section 2.3.6.2.5.8
engine_fail	Section 2.3.6.2.5.6

Table 2.3-360: engine_clog_fuel_filter Information.

2.3.6.2.6.4 engine_fix_fuel_filter

This rouitne unsets the status and control indicators from fuel filter clogged. The engine starter and engine are repaired.

Calls	
Function	Where Described
controls_fuel_filter_normal	Section 2.3.2
engine_starter_fix	Section 2.3.6.2.5.9
engine_fix	Section 2.3.6.2.5.7

Table 2.3-361: engine_fix_fuel_filter Information.

2.3.6.2.6.5 engine_clog_air_filter

This routine sets control indicators and status to air filter clogged. The engine starter and engine are failed.

Calls	
Function	Where Described
controls_air_cleaner_clogged	Section 2.3.2
engine_starter_fail	Section 2.3.6.2.5.8
engine_fail	Section 2.3.6.2.5.6

Table 2.3-362: engine_clog_air_filter Information.

2.3.6.2.6.6 engine_fix_air_filter

This rouitne unsets the status and control indicators from air filter clogged. The engine starter and engine are repaired.

Calls	
Function	Where Described
controls_air_cleaner_normal	Section 2.3.2
engine_starter_fix	Section 2.3.6.2.5.9
engine_fix	Section 2.3.6.2.5.7

Table 2.3-363: engine_fix_air_filter Information.

2.3.6.2.6.7 engine_oil_leak

This routine simulates an oil leak. The engine starter and engine are failed.

Calls	
Function	Where Described
engine_starter_fail	Section 2.3.6.2.5.8
engine_fail	Section 2.3.6.2.5.6

Table 2.3-364: engine_oil_leak Information.

2.3.6.2.6.8 engine_oil_normal

This routine repairs an oil leak. The engine starter and engine are repaired.

Calls	
Function	**Where Described**
engine_starter_fix	Section 2.3.6.2.5.9
engine_fix	Section 2.3.6.2.5.7

Table 2.3-365: engine_oil_normal Information.

2.3.6.2.6.9 engine_fail_starter

This routine sets the engine starter status to fail. The engine starter is failed.

Calls	
Function	**Where Described**
engine_starter_fail	Section 2.3.6.2.5.8

Table 2.3-366: engine_fail_starter Information.

2.3.6.2.6.10 engine_fix_starter

This routine repairs the engine starter, resetting its status to normal.

Calls	
Function	**Where Described**
engine_starter_fix	Section 2.3.6.2.5.9

Table 2.3-367: engine_fix_starter Information.

2.3.6.2.6.11 engine_failure_update

This routine increments the appropriate failure timer if the status of an engine subsystem is fail.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
fP	pointer to FAIL_STAT	Section 2.3.6.2.6

Table 2.3-368: engine_failure_update Information.

2.3.6.2.6.12 engine_init_power

This routine sets the initial power level of the engine to *val*, which is passed as an argument.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
val	REAL	sim_types.h

Table 2.3-369: engine_init_power Information.

2.3.6.2.6.13 engine_power_loss

This routine calculates the reduction in the power level due to a clogged oil filter, clogged air filter, leaking fuel, and leaking coolant. The reduced power level value is returned.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
power_percent	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
initial_power_percent * power_percent	REAL	The reduced power level.
Calls		
Function	**Where Described**	
min	sim_macros.h	
max	sim_macros.h	
engine_off	Section 2.3.6.2.5.18	

Table 2.3-370: engine_power_loss Information.

2.3.6.2.6.14 check_starter_failure

This routine checks whether there is a failure of the air filter, fuel filter, coolant, oil, or starter.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	Air filter or fuel filter or coolant or oil or starter is failed.
FALSE	int	None of the above is failed.

Table 2.3-371: check_starter_failure Information.

2.3.6.2.6.15 check_engine_failure

This routine checks whether there is a failure of the air filter, fuel filter, coolant, or oil.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	Air filter or fuel filter or coolant or oil is failed.
FALSE	int	None of the above is failed.

Table 2.3-372: check_engine_failure Information.

2.3.6.3 Vehicle Subsystems

The CSU's required to simulate the M2's subsystems are as follows.

m2_elecsys.c
m2_vision.c
m2_isu.c

2.3.6.3.1 m2_elecsys.c
(./simnet/release/src/vehicle/m2/src/m2_elecsys.c [m2_elecsys.c])

The model of the M2's electrical system is based the assumption that charge and voltage are linearly related and that the battery charges linearly over time. While many small components (such as indicator lamps) consume electrical charge, only the major charge consumers are modeled. The major charge consumers consist of the starter, TOW missile, 25 mm gun, turret drive system, fuel transfer pump, radio, and intercom for the M2. Routines to support this model are found in m2_elecsys.c.

When the engine is running, it is assumed that all electrical components are obtaining electrical charge for the alternator/generator, which is modeled as an infinite amount of charge. As long as the engine is running and the alternator/ generator has not failed, all requests for electrical charge are granted. The alternator/generator will also charge up a battery which is less than fully charged. However, if the engine is off or the alternator/generator has failed, the electrical components must obtain electrical charge from the battery.

Each of the modeled electrical components requires a specific amount of electrical charge to be present. When the component is used, the electrical system is queried to see if there is enough charge present to accommodate the request. If so, the stored charge is depleted by the amount requested.

The following assumptions are made in modeling the electrical system:

- Generator is 220 amps, regulated to 24VDC
- Hull Batteries: 4 each, 100amp-hr each, consisting of wet cells connected in a series/parallel arrangement for a total: 24v, 200amp-hr.
- The hull is a ground.

- The turret receives power from the hull power distribution system.

- The voltage gage indicates the following battery and generator conditions:
  left red zone - low battery charge with engine off. Battery may not start the engine.
  yellow zone - indicates normal battery voltage with engine off. Indicates generator not charging with engine running.
  green zone - indicates generator charging normally with engine running.
  right red zone- indicates generator overcharging with engine running.

- Turning the Master power off before the engine has stopped can damage the electrical system.

- The turret electrical system:
  Operates the 25mm gun, the tow launcher, and the turret.
  The tow, 12mm gun, and turret will no work if the battery has less than 18.5v (LEFT_RED_ZONE_MAX)
  It takes approximately 90 minutes to recharge the TURRET_EMERGENCY_BATTERIES from LEFT_RED_ZONE_MAX to TURRET_MAX_CHARGE;

- The gunner's handles control the turret electrically

- Drive system power allows commander to override gunner's control of the turret:
  Four 12 volt wet cell batteries located in hull
  Two 12 volt wet cell batteries located in turret
  Total: 24v, 100amp - hr.
  Stabilization system is electric
  25mm gun -> 1.5hp (not an assumption)

- Due to use of the radio:
  Radio set operating power: 22 - 30VDC
  The radio can run continuously with 22v input
  The radio can run for 1 hour with 30v input.

- If the engine accessory switch is off, the pump between the bottom and top fuel tanks is off.

- With the generator charging properly, the meter should remain fully within the GREEN ZONE. This is modeled linearly as in the M1.

- The tow, 12mm gun, and turret will not operate if the battery has less than 18.5v (LEFT_RED_ZONE_MAX) . TOW_DISCHARGE_RATE = (TURR_MAX_CHARGE - LEFT_RED_ZONE_MAX) * TURR_V_TO_Q)/2 due to the fact that the TOW can be fired twice with the engine off.

- The hull battery is discharged to STARTER_MIN_VOLTAGE (17.5V) after four 30 second starts. Therefore, the discharge rate per tick is:
  STARTER_DISCHARGE_RATE = (HULL_MAX_VOLTAGE - STARTER_MIN_VOLTAGE* HULL_V_TO_Q / (4 * 30sec * 15 ticks/sec)
  = (24v - 17.5v) * 200amp-hr/24v / 1800 ticks
  = 0.03009259259 amp-hr/tick

- The turret elevation drive has an alloted power of 32 hp (an assumption).
  1 hp = 746 watts, therefore 32hp = 23872 watts.
  power = vi, therefore current = 23872 watts/24 volts.
  i = dq/dt
    dt = 1/15 sec/frame * 1/60 min/sec * 1/60 hr/min = 1/54000 hr/tick
    dq = 23872/24 * 1/54000 = 23872/1296000 amp-hr/tick for linear discharge.
  = 0.0008634259 amp-hr/tick
  = TURRET_ELEV_DISCHARGE_RATE
  The same value is currently used for the TURRET_TRAVERSE_DISCHARGE_RATE.

- The 25mm gun has an alloted power of 1.5hp (not an assumption).
  1 hp = 746 watts -> 1.5hp = 1119 watts
  power = vi -> current = 1119 watts/24 volts
  i = dq/dt
    dt = 1/15 sec/frame * 1/60 min/sec * 1/60 hr/min = 1/54000 hr/tick
    dq = 1119/24 * 1/54000 = 1119/1296000 amp-hr/tick for linear discharge
  = 0.01841975 amp-hr/tick = GUN_25MM_DISCHARGE_RATE

- It takes approximately 90 minutes to recharge the TURRET_BACKUP_BATTERIES from LEFT_RED_ZONE_MAX to TURRET_MAX_CHARGE. The recharge rate is:
  (TURRET_MAX_CHARGE - LEFT_RED_ZONE_MAX)/ (90 min * 60sec/min * 15ticks/sec)

- The BATTERY_RECHARGE_RATE = 0.0002829218 amp-hr/tick.

- If the generator fails while the engine is running, the engine will run for approximately 45 minutes. The hull battery charge goes from HULL_MAX_VOLTAGE to LEFT_RED_ZONE_MIN in 45 minutes (40500 ticks) due to the running engine.
  ENGINE_DISCHARGE_RATE = (HULL_MAX_VOLTAGE - LEFT_RED_ZONE_MIN) / 40500

Includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libfail.h"
"failure.h"
"m2_meter.h"
"m2_engine.h"

"m2_cntrl.h"
"m2_elecsys.h"
"m2_main.h"

The following conversion factors are defined:
HULL_MAX_CHARGE
HULL_MAX_VOLTAGE 24.0
HULL_V_TO_Q
HULL_Q_TO_V
TURR_MAX_CHARGE
TURR_MAX_VOLTAGE
TURR_V_TO_Q
TURR_Q_TO_V

The following zones on the meter are defined:
LEFT_RED_ZONE_MIN
LEFT_RED_ZONE_MAX
YELLOW_ZONE_MAX
GREEN_ZONE_MAX
RIGHT_RED_ZONE_MAX
STARTER_MIN_VOLTAGE - approximately 1/4 into left red zone
STARTER_MIN_CHARGE
TURR_BATTERY_MIN

The following engine parameters are defined:
IDLE
MAX_RADS
SLOPE
MIN_RADS - if less than this value, the engine is off

The following Boolean values for battery states are defined:
NEW
DEAD - the battery holds no charge

Additional Defines:
TOW_DISCHARGE_RATE
STARTER_DISCHARGE_RATE
TURRET_ELEV_DISCHARGE_RATE
TURRET_TRAVERSE_DISCHARGE_RATE
GUN_25MM_DISCHARGE_RATE
BATTERY_RECHARGE_RATE
ENGINE_DISCHARGE_RATE
RADIO_DISCHARGE_RATE
INTERCOM_DISCHARGE_RATE
FUEL_XFER_PUMP_DISCHARGE_RATE

The following are declared:
hull_power_status - ON or OFF
turret_power_status - ON or OFF
turret_backup_power_status - ON or OFF
hull_battery_charge
turret_battery_charge
turret_backup_battery_charge
hull_battery_status - NEW or DEAD

turret_backup_battery_status - NEW or DEAD
generator_status - WORKING or BROKEN
drive_malfunction_status - TRUE or FALSE
gun_25mm_malfunction_status - TRUE or FALSE
tow_circuit_open_status - TRUE or FALSE
turret_electsys_dead
hull_electsys_dead
voltmeter_enabled - TRUE or FALSE

2.3.6.3.1.1 electsys_charge_battery

This routine recharges the hull and turret batteries.

Calls	
Function	**Where Described**
min	sim_macros.h
controls_hull_electsys_reborn	Section 2.3.2
controls_turret_backup_electsys_reborn	Section 2.3.2

Table 2.3-373: electsys_charge_battery Information.

2.3.6.3.1.2 electsys_discharge_hull_battery

This routine discharges the battery by *delta* if the engine is off.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
delta	REAL	sim_types.h
Calls		
Function	**Where Described**	
max (macro definition)	sim_macros.h	
fail_break_system	Section 2.5.4.8.1	

Table 2.3-374: electsys_discharge_hull_battery Information.

2.3.6.3.1.3 electsys_discharge_turret_backup_battery

This routine discharges the turret backup battery by *delta* if the engine is off.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
delta	REAL	sim_types.h
Calls		
Function	**Where Described**	
max (macro definition)	sim_macros.h	
fail_break_system	Section 2.5.4.8.1	

Table 2.3-375: electsys_discharge_turret_backup_battery Information.

2.3.6.3.1.4 electsys_rads_to_volts

This routine allows the electrical system meter to reflect the change in rpms of the engine. It is only used with the engine on. This is accomplished by the y = mx + b formula where y is volts and x is rads per sec. The y values are the entire GREEN ZONE of the meter.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
rads	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
YELLOW_ZONE_MAX	static REAL	voltmeter value
GREEN_ZONE_MAX	static REAL	voltmeter value
SLOPE*(rads-IDLE) + HULL_MAX_VOLTAGE	static REAL	voltmeter value

Table 2.3-376: electsys_rads_to_volts Information.

2.3.6.3.1.5 electsys_handle_leaky_hull_battery

This routine handles the case of a leaky battery. The battery charges up, but when the load is applied, the voltage immediately drops to WEAKLY_CHARGED. The battery charge drops to *limit* after the load has been applied. It corresponds to the value of the Left Red Zone converted to charge.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
Limit	REAL	sim_types.h
Calls		
Function	**Where Described**	
electsys_turret_backup_power_request	Section 2.3.6.3.1.7	
controls_hull_electsys_dead	Section 2.3.2	

Table 2.3-377: electsys_handle_leaky_hull_battery Information.

2.3.6.3.1.6 electsys_handle_leaky_turret_backup_battery

This routine handles the case of a leaky turret backup battery. The battery charge drops to *limit* after the load has been applied. It corresponds to the value of the Left Red Zone converted to charge.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
Limit	REAL	sim_types.h
Calls		
Function	**Where Described**	
controls_turret_hackup_electsys_dead	Section 2.3.2	

Table 2.1-378: electsys_handle_leaky_turret_backup_battery Information.

2.3.6.3.1.7 electsys_turret_backup_power_request

This routine is called to make sure the turret backup power can be turned on if the hull power is not on.

Return Values		
Return Value	**Type**	**Meaning**
turret_backup_power_status	static BOOLEAN	status of the turret backup power, either ON or OFF
Calls		
Function	**Where Described**	
controls_commander_backup_power_on	Section 2.3.2	
controls_commander_backup_power_off	Section 2.3.2	

Table 2.3-379: electsys_turret_backup_power_request Information.

2.3.6.3.1.8 electsys_simul

If the engine is running and generator_status is true, then the engine is running off the generator rather than the battery. When the engine is running with the generator, the battery is recharged. The battery discharges when the engine is running off the battery and when the engine is off.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
engine_rads	REAL	sim_types.h
Calls		
Function	**Where Described**	
engine_speed	Section 2.3.6.2.5.11	
meter_volt_set	Section 2.3.2.3.3	
engine_running	Section 2.3.6.2.5.10	
electsys_charge_battery	Section 2.3.6.3.1.1	
electsys_discharge_hull_battery	Section 2.3.6.3.1.2	

Table 2.3-380: electsys_simul Information.

2.3.6.3.1.9 electsys_hull_dead

This routine sets the hull electrical system to the dead state. *hull_electsys_dead* is set to TRUE, and *hull_power_status* is set to OFF.

Calls	
Function	**Where Described**
controls_hull_electsys_dead	Section 2.3.2

Table 2.3-381: electsys_hull_dead Information.

2.3.6.3.1.10 electsys_turret_dead

This routine sets the turret electrical system to the dead state. *turret_electsys_dead* is set to TRUE, and *turret_backup_power_status* is set to OFF.

Calls	
Function	**Where Described**
controls_turret_backup_electsys_dead	Section 2.3.2

Table 2.3-382: electsys_turret_dead Information.

2.3.6.3.1.11 electsys_dead

This routine calls routines which set the hull and turret electrical systems to the dead state.

Calls	
Function	**Where Described**
electsys_hull_dead	Section 2.3.6.3.1.9
electsys_turret_dead	Section 2.3.6.3.1.10

Table 2.3-383: electsys_dead Information.

2.3.6.3.1.12 electsys_drive_malfunction_status

This routine returns the drive malfunction status.

Return Values		
Return Value	**Type**	**Meaning**
drive_malfunction_status	BOOLEAN	drive malfunction status TRUE - malfunctioning FALSE - functioning

Table 2.3-384: electsys_drive_malfunction_status Information.

2.3.6.3.1.13 electsys_set_turret_drive_status

This routine sets the turret drive status to *status*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
status	BOOLEAN	sim_types.h

Table 2.3-385: electsys_set_turret_drive_status Information.

2.3.6.3.1.14 electsys_25mm_gun_malfunction_status

This routine returns the 25mm gun malfunction status.

Return Values		
Return Value	**Type**	**Meaning**
gun_25mm_malfunction_stat us	BOOLEAN	25mm gun malfunction status TRUE - malfunctioning FALSE - functioning

Table 2.3-386: electsys_25mm_gun_malfunction_status Information.

2.3.6.3.1.15 electsys_set_25mm_gun_malfunction_status

This routine sets the 25mm gun malfunction status to *status*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
status	BOOLEAN	sim_types.h

Table 2.3-387: electsys_set_25mm_gun_malfunction_status Information.

2.3.6.3.1.16 electsys_tow_circuit_open_status

This routine returns the tow circuit open status.

Return Values		
Return Value	**Type**	**Meaning**
tow_circuit_open_status	BOOLEAN	status of tow circuit TRUE - open circuit FALSE - closed circuit

Table 2.3-388: electsys_tow_circuit_open_status Information.

2.3.6.3.1.17 electsys_set_tow_circuit_open_status

This routine sets the tow circuit open status to *status*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
status	BOOLEAN	the tow circuit open status

Table 2.3-389: electsys_set_tow_circuit_open_status Information.

2.3.6.3.1.18 electsys_hull_power_request

This routine is called by the controls routines to make sure the hull power can be turned on.

Return Values		
Return Value	**Type**	**Meaning**
hull_power_status	BOOLEAN	the status of hull power
Calls		
Function	**Where Described**	
engine_running	Section 2.3.6.2.5.10	
controls_commander_backup_power_off	Section 2.3.2	

Table 2.3-390: electsys_hull_power_request Information.

2.3.6.3.1.19 electsys_turret_power_request

This routine is called by the controls routines to make sure the turret power can be turned on.

Return Values		
Return Value	**Type**	**Meaning**
turret_power_status = ON	BOOLEAN	turret power is on and hull power is on
electsys_turret_backup_power_request()	BOOLEAN	is backup power available?
Calls		
Function	**Where Described**	
electsys_turret_backup_power_request	Section 2.3.6.3.1.7	

Table 2.3-391: electsys_turret_power_request Information.

2.3.6.3.1.20 electsys_engine_start_request

This routine determines if the battery is sufficiently charged for the engine to be started and determines the charge by the STARTER_DISCHARGE_RATE.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	engine can be started
FALSE	BOOLEAN	engine cannot be started

Calls	
Function	**Where Described**
electsys_discharge_hull_battery	Section 2.3.6.3.1.2

Table 2.3-392: electsys_engine_start_request Information.

2.3.6.3.1.21 electsys_tow_request

This routine determines if the Tow launcher can be operated. If it can, the routine returns TRUE. If the TOW launcher is not working or if there is insufficient power for TOW operation, the routine returns FALSE.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	engine is running or sufficient battery power for operation is available
FALSE	BOOLEAN	Tow circuit open, TOW not working, or insufficient power available

Calls	
Function	**Where Described**
electsys_discharge_hull_battery	Section 2.3.6.3.1.2
electsys_discharge_turret_backup_battery	Section 2.3.6.3.1.3

Table 2.3-393: electsys_tow_request Information.

2.3.6.3.1.22 electsys_turret_elevation_request

This routine determines if the turret elevation can be adjusted. If the drive is broken or if there is insufficient power available, the elevation cannot be adjusted and the routine returns FALSE. Otherwise, the elevation can be adjusted and the routine returns TRUE. *percent* is the percent of maximum slew rate (handle+stab displacement).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
percent	REAL	sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	elevation can be adjusted
FALSE	BOOLEAN	elevation cannot be adjusted - drive is broken - power insufficient or unavailable

Calls	
Function	**Where Described**
engine_running	Section 2.3.6.2.5.10
electsys_discharge_hull_battery	Section 2.3.6.3.1.2
electsys-discharge_turret_backup_battery	Section 2.3.6.3.1.3

Table 2.3-394: electsys_turret_elevation_request Information.

2.3.6.3.1.23 electsys_turret_traverse_request

This routine determines if the turret traverse position can be adjusted. If the drive is broken or if there is insufficient power available, the traverse position cannot be adjusted and the routine returns FALSE. Otherwise, the traverse position can be adjusted and the routine returns TRUE. *percent* is the percent of maximum slew rate (handle+stab displacement).

Parameters		
Parameter	**Type**	**Where Typedef Declared**
percent	REAL	sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	traverse position can be adjusted
FALSE	BOOLEAN	traverse position cannot be adjusted - drive is broken - power insufficient or unavailable
Calls		
Function	**Where Described**	
engine_running	Section 2.3.6.2.5.10	
electsys_discharge_hull_battery	Section 2.3.6.3.1.2	
electsys_discharge_turret_backup_battery	Section 2.3.6.3.1.3	

Table 2.3-395: electsys_turret_transverse_request Information.

2.3.6.3.1.24 electsys_25mm_gun_request

This routine determines if the 25mm gun can be operated. It returns TRUE if the gun can be operated and retuns FALSE otherwise.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	can be operated
FALSE	BOOLEAN	cannot be operated
Calls		
Function	**Where Described**	
engine_running	Section 2.3.6.2.5.10	
electsys_discharge_hull_battery	Section 2.3 6.3.1.2	
electsys-discharge_turret_backup_battery	Section 2.3.6.3.1.3	

Table 2.3-396: electsys_25mm_gun_request Information.

2.3.6.3.1.25 electsys_fuel_xfer_pump_request

This routine determines if the fuel transfer pump can be operated. If the engine is running or if there is sufficient battery power available, the pump will operate, and the routine will return TRUE. If there is not sufficient power available, the pump will not run and the routine returns FALSE.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	BOOLEAN	pump will operate
FALSE	BOOLEAN	pump will not operate

Calls	
Function	**Where Described**
engine_running	Section 2.3.6.2.5.10
electsys_discharge_hull_battery	Section 2.3.6.3.1.2

Table 2.3-397: electsys_fuel_xfer_pump_request Information.

2.3.6.3.1.26 electsys_get_hull_battery_voltage

This routine returns the current hull battery voltage.

Return Values		
Return Value	**Type**	**Meaning**
hull_battery_charge*HULL_Q_TO_V	REAL	the current hull battery voltage

Table 2.3-398: electsys_get_hull_battery_voltage Information.

2.3.6.3.1.27 electsys_get_turret_backup_battery_voltage

This routine returns the current turret backup battery voltage.

Return Values		
Return Value	**Type**	**Meaning**
turret_backup_battery_charge*TURR_Q_TO_V	REAL	the current turret backup battery voltage

Table 2.3-399: electsys_get_turret_backup_battery_voltage Information.

2.3.6.3.1.28 electsys_replace_generator

This routine sets the generator status to WORKING.

2.3.6.3.1.29 electsys_generator_failure

This routine sets *generator_status* to BROKEN. When the generator fails, the engine runs off the battery, which lasts approximately 45 minutes.

2.3.6.3.1.30 electsys_replace_hull_battery

This routine replaces a dead hull battery.

Calls	
Function	**Where Described**
controls_hull_electsys_reborn	Section 2.3.2

Table 2.3-400: electsys_replace_hull_battery Information.

2.3.6.3.1.31 electsys_replace_turret_backup_battery

This routine replaces a dead turret backup battery.

Calls	
Function	**Where Described**
controls_turret_backup_electsys_reborn	Section 2.3.2

Table 2.3-401: electsys_replace_turret_backup_battery Information.

2.3.6.3.1.32 electsys_turret_power_off

This routine turns the turret power off by setting *turret_power_status* to OFF and setting *turret_backup_power_status* to OFF.

2.3.6.3.1.33 electsys_hull_power_off

This routine turns the hull power off by setting *hull_power_status* to OFF. Turret backup power is requested.

Calls	
Function	**Where Described**
electsys_turret_backup_power_request	Section 2.3.6.3.1.7

Table 2.3-402: electsys_hull_power_off Information.

2.3.6.3.1.34 print_electsys_variables

The following data is printed:

hull_battery_charge
turret_backup_battery_charge
hull_power_status
turret_power_status
turret_backup_power_status
hull_electsys_dead
turret_electsys_dead

2.3.6.3.1.35 electsys_reborn

This routine reactivates the M2's electrical system.

Calls	
Function	**Where Described**
electsys_hull_reborn	Section 2.3.6.3.1.37
electsys_turret_reborn	Section 2.3.6.3.1.36

Table 2.3-403: electsys_reborn Information.

2.3.6.3.1.36 electsys_turret_reborn

This routine reactivates the turret's electrical system.

Calls	
Function	**Where Described**
controls_turret_backup_elect sys_reborn	Section 2.3.2

Table 2.3-404: electsys_turret_reborn Information.

2.3.6.3.1.37 electsys_hull_reborn

This routine reactivates the hull's electrical system.

Calls	
Function	**Where Described**
controls_hull_electsys_rebor n	Section 2.3.2

Table 2.3-405: electsys_hull_reborn Information.

2.3.6.3.1.38 electsys_init_batteries

This routine initializes the hull and turret backup batteries.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
hull	REAL	sim_types.h
turret	REAL	sim_types.h

Table 2.3-406: electsys_init_batteries Information.

2.3.6.3.1.39 electsys_voltmeter_disabled

This routine disables a voltmeter by setting *voltmeter_enabled* to FALSE.

2.3.6.3.1.40 electsys_init

This routine initializes the M2's electrical system.

Calls	
Function	**Where Described**
meter_volt_set	Section 2.3.2.3.3
fail_init_failure	Section 2.5.4.11.3

Table 2.3-407: electsys_init Information.

2.3.6.3.2 m2_vision.c
(./simnet/release/src/vehicle/m2/src/m2_vision.c [m2_vision.c])

The routines which control the state of the individual viewports are found in m2_vision.c. These routines break the viewports (turn them off) and fix them by toggling bits in libvflags. The ability of the cupolas to pitch up and down is also controlled in the vision files. Furthermore, on the M2, the gunner's unity vision block and the commander's right vision block both change resolution and shape to become the gunner's isu and gunner's isu extension respectively.

You must send a view mode message any time you want to change the screen resolution (cmdr_rt/gnr_isu_ext).

Branch indices and branch masks have the following meaning:

branch index	branch mask	purpose	
0	4	cmdr rt ISU/standard	T=ISU
2	1	cmdr rt ISU/standard	T=ISU
1	1	cmdr rt ISU 4x/12x	T=4x
0	1	cmdr pitch up/oth?	T=up
0	2	cmdr pitch dn/none?	T=down
1	2	gnr ISU/standard	T=ISU
3	1	gnr ISU/standard	T=ISU
1	1	gnr 4x/12x	T=4x

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"failure.h"
"libvflags.h"
"libfail.h"
"m2_vision.h"

Defines:

Symbol	Value
GNR_MAG	0x0001
CMDR_BR_VAL	0
CMDR_ISU_VAL	2
GNR_ISU_VAL	3
ISU_MAG_VAL	1
ISU_BIT	0x0001
TC_DEFAULT	0x34ff
TC_GNR_ISU	0x8400
TC_CMDR_ISU	0x4400
SKY_ZERO	0
SKY_ONE	1
SKY_TWO	2
SKY_THREE	3

DVRS_FAR_LT	VIEWP_5	
DVRS_LT	VIEWP_7	
DVRS_CTR	VIEWP_6	
DVRS_RT	VIEWP_1	
CMDR_LT	VIEWP_4	
CMDR_CTR	VIEWP_3	
CMDR_RT	VIEWP_0	ISU ext
GNR_VIEWP	VIEWP_2	ISU
DVRS_VIEWP	(DVRS_FAR_LT | DVRS_LT | DVRS_CRT | DVRS_RT)	
CMDRS_VIEWP	(CMDR_LT | CMDR_CTR | CMDR_RT)	
GNR_ISU_WORKING	GNR_VIEWP	
GNR_WORKING	GNR_VIEWP	
CMDR_ISU_WORKING	CMDR_RT	
CMDR_WORKING	CMDR_RT	
GNR_ISU_BROKEN	0x0	
GNE_BROKEN	0x0	
CMDR_ISU_BROKEN	0x0	
CMDR_BROKEN	0x0	
VISION_BLOCK_SELF_REPAIR_TIME	10	minutes
ISU_SELECTED	1	
ISU_DESELECTED	0	
VM_CMR_RT	0	definitions for configuration viewports and view modes
VM_GNR	1	
VM_CMR_CTR	2	
VM_CMR_LT	3	
VM_CVR_RT	4	
VM_CVR_FLT	5	
VM_DVR_CTR	6	
VM_DVR_LT	7	

int declarations and initialization:

sky_color
brow_pad_changed = FALSE
cmdr_isu_status = CMDR_ISU_WORKING
cmdr_status = CMDR_WORKING
gunner_isu_status = GNR_ISU_WORKING
gunner_status = GNR_WORKING
gunner_state = ISU_DESELECTED
cmdr_state = ISU_DESELECTED

Procedure declarations:

fit_engine_torque()
engine_run()
engine_crank()
engine_off()

REAL declarations:

engine_speed_fraction

2.3.6.3.2.1 vision_get_sky_color

This routine returns the sky color.

Return Values		
Return Value	Type	Meaning
sky_color	int	The sky color

Table 2.3-408: vision_get_sky_color Information.

2.3.6.3.2.2 vision_toggle_sky_color

This routine toggles the sky color.

2.3.6.3.2.3 cig_gps_mag_12x

This routine switches the gunner's primary sight unit to the magnification of 12X.

Calls	
Function	Where Described
clr_br_bit	Section 2.1.2.2.4.1.1
clear_view_flags	Section 2.1.2.2.4.2.1
set_br_bit	Section 2.1.2.2.4.6.1

Table 2.3-409: cig_gps_mag_12x Information.

2.3.6.3.2.4 cig_gps_mag_4x

This routine switches the gunner's primary sight unit to the magnification of 4X.

Calls	
Function	Where Described
set_br_bit	Section 2.1.2.2.4.6.1
clear_view_flags	Section 2.1.2.2.4.2.1

Table 2.3-410: cig_gps_mag_4x Information.

2.3.6.3.2.5 vision_cmdrs_pitch_up

This routine pitches up the commander's vision.

Calls	
Function	Where Described
set_br_vals	Section 2.1.2.2.4.7.1
clear_view_flags	Section 2.1.2.2.4.2.1
set_br_bit	Section 2.1.2.2.4.6.1

Table 2.3-411: vision_cmdrs_pitch_up Information.

2.3.6.3.2.6 vision_cmdrs_pitch_ahead

This routine pitches ahead the commander's vision.

Calls	
Function	**Where Described**
set_br_vals	Section 2.1.2.2.4.7.1
clear_view_flags	Section 2.1.2.2.4.2.1
set_br_bit	Section 2.1.2.2.4.6.1

Table 2.3-412: vision_cmdrs_pitch_ahead Information.

2.3.6.3.2.7 vision_cmdrs_pitch_down

This routine pitches down the commander's vision.

Calls	
Function	**Where Described**
set_br_vals	Section 2.1.2.2.4.7.1
clear_view_flags	Section 2.1.2.2.4.2.1
set_br_bit	Section 2.1.2.2.4.6.1

Table 2.3-413: vision_cmdrs_pitch_down Information.

2.3.6.3.2.8 vision_restore_all_blocks

This routine sets view flags and view modes to restore vision on all the screens.

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

Calls	
Function	**Where Described**
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.3-414: vision_restore_all_blocks Information.

2.3.6.3.2.9 vision_break_all_blocks

This routine clears view flag bits for all views, blackening all the screens when catastrophic kill occurs.

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

Calls	
Function	**Where Described**
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.3-415: vision_break_all_blocks Information.

2.3.6.3.2.10 vision_break_isu

This routine clears view flag bits, blackening the screen representing the integrated sight unit (isu) of the gunner.

Errors	
Error Name	**Reason for Error**
unknown gunner state ...	The variable gunner_state has an unexpected value.

Calls	
Function	**Where Described**
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.3-416: vision_break_isu Information.

2.3.6.3.2.11 vision_break_isu_ext

This routine clears view flag bits, blackening the screen representing the integrated sight unit extension of the commander.

Errors	
Error Name	**Reason for Error**
unknown cmdr state ...	The variable cmdr_state has an unexpected value.
Calls	
Function	**Where Described**
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.3-417: vision_break_isu_ext Information.

2.3.6.3.2.12 vision_break_driver_blocks

This routine clears view flag and view mode bits, blackening the screen used by the driver.

Calls	
Function	**Where Described**
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.3-418: vision_break_driver_blocks Information.

2.3.6.3.2.13 vision_break_cmdrs_blocks

This routine clears view flag and view mode bits, blackening the screen used by the commander.

Errors	
Error Name	**Reason for Error**
unknown cmdr state ...	The variable cmdr_state has an unexpected value.
Calls	
Function	**Where Described**
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.3-419: vision_break_cmdrs_blocks Information.

2.3.6.3.2.14 vision_restore_isu

This routine sets view flag and view mode bits, restoring the view on the screens used for the integrated sight unit of the gunner.

Errors	
Error Name	**Reason for Error**
unknown gunner state ...	The variable gunner_state has an unexpected value.
Calls	
Function	**Where Described**
set_view_flags	Section 2.1.2.2.4.8.1
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.3-420: vision_restore_isu Information.

2.3.6.3.2.15 vision_restore_isu_ext

This routine sets view flag and view mode bits, restoring the view on the screens used for the integrated sight unit extension of the commander.

Errors	
Error Name	**Reason for Error**
unknown cmdr state ...	The variable cmdr_state has an unexpected value.
Calls	
Function	**Where Described**
set_view_flags	Section 2.1.2.2.4.8.1
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.3-421: vision_restore_isu_ext Information.

2.3.6.3.2.16 vision_restore_driver_blocks

This routine sets view flags and view modes, restoring the view on the screen used by the driver.

Calls	
Function	**Where Described**
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.3-422: vision_restore_driver_blocks Information.

2.3.6.3.2.17 vision_restore_cmdrs_blocks

This routine sets view flag and view modes, restoring the view on the screens used by the commander.

Errors	
Error Name	**Reason for Error**
unknown cmdr state ...	The variable cmdr_state has an unexpected value.
Calls	
Function	**Where Described**
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.3-423: vision_restore_cmdrs_blocks Information.

2.3.6.3.2.18 vision_break_gunners_block

This routine clears view flag and view mode bits, blackening the screens used by the gunner.

Errors	
Error Name	**Reason for Error**
unknown gunner state ...	The variable gunner_state has an unexpected value.
Calls	
Function	**Where Described**
set_view_flags	Section 2.1.2.2.4.8.1
clear_view_flags	Section 2.1.2.2.4.2.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.3-424: vision_break_gunners_block Information.

2.3.6.3.2.19 vision_restore_gunners_block

This routine sets view flag and view mode bits, restoring the view on the screens used by the gunner.

Errors	
Error Name	**Reason for Error**
unknown gunner state ...	The variable gunner_state has an unexpected value.
Calls	
Function	**Where Described**
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1

Table 2.3-425: vision_restore_gunners_block Information.

2.3.6.3.2.20 vision_gunner_brow_pad_on

This routine enables the integrated sight unit view of the gunner.

Calls	
Function	**Where Described**
set_br_bit	Section 2.1.2.2.4.6.1
clear_view_flags	Section 2.1.2.2.4.2.1
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1
controls_gunner_brow_pad_on	Section 2.3.2

Table 2.3-426: vision_gunner_brow_pad_on Information.

2.3.6.3.2.21 vision_gunner_brow_pad_off

This routine disables the integrated sight unit view of the gunner, enabling the unity vision view of the gunner.

Calls	
Function	**Where Described**
clr_br_bit	Section 2.1.2.2.4.1.1
clear_view_flags	Section 2.1.2.2.4.2.1
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1
controls_gunner_brow_pad_off	Section 2.3.2

Table 2.3-427: vision_gunner_brow_pad_off Information.

2.3.6.3.2.22 vision_commander_brow_pad_on

This routine enables the integrated sight unit extension view of the commander.

Calls	
Function	**Where Described**
set_br_bit	Section 2.1.2.2.4.6.1
clear_view_flags	Section 2.1.2.2.4.2.1
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1
controls_commander_brow_pad_on	Section 2.3.2

Table 2.3-428: vision_commander_brow_pad_on Information.

2.3.6.3.2.23 vision_commander_brow_pad_off

This routine disables the integrated sight unit extension view of the commander.

Calls	
Function	**Where Described**
clr_br_bit	Section 2.1.2.2.4.1.1
clear_view_flags	Section 2.1.2.2.4.2.1
set_view_flags	Section 2.1.2.2.4.8.1
set_vmodes	Section 2.1.2.2.4.9.1
controls_commander_brow_pad_off	Section 2.3.2

Table 2.3-429: vision_commander_brow_pad_off Information.

2.3.6.3.2.24 print_br_values

This routine displays branch values.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
wd	pointer to WORD	mass_stdc.h
vf	WORD	mass_stdc.h

Calls	
Function	**Where Described**
get_br_vals	Section 2.1.2.2.4.3.1
get_view_flags	Section 2.1.2.2.4.4.1

Table 2.3-430: print_br_values Information.

2.3.6.3.2.25 get_cmdr_state

This routine returns the commander's view state.

Return Values		
Return Value	**Type**	**Meaning**
cmdr_state	int	Normal end.

Table 2.3-431: get_cmdr_state Information.

2.3.6.3.2.26 get_gunner_state

This routine returns the gunner's view state.

Return Values		
Return Value	**Type**	**Meaning**
gunner_state	int	Normal end.

Table 2.3-432: get_gunner_state Information.

2.3.6.3.2.27 get_brow_pad_status

This routine returns the brow pad status.

Return Values		
Return Value	**Type**	**Meaning**
FALSE	int	Brow pad not changed.
TRUE	int	Brow pad changed.

Table 2.3-433: get_brow_pad_status Information.

2.3.6.3.2.28 vision_init

This routine initializes software for all the vision blocks.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
Calls		
Function	**Where Described**	
clear_view_flags	Section 2.1.2.2.4.2.1	
set_view_flags	Section 2.1.2.2.4.8.1	
set_vmodes	Section 2.1.2.2.4.9.1	
fail_init_failure	Section 2.5.4.11.2	

Table 2.3-434: vision_init Information.

2.3.6.3.3 m2_isu.c

(./simnet/release/src/vehicle/m2/src/m2_isu.c [m2_isu.c])

The m2_isu.c CSU determines which reticle (hi mag, lo mag, TOW) is displayed in the gunner's isu and gunner's isu extension viewports.

Includes:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"m2_turr_mx.h"
"m2_cntrl.h"
"m2_isu.h"
"pro_sim.h"
"mun_type.h"

Defines:

Symbols	Value
LO_MAG_RET	0
HI_MAG_RET	1
TOW_RET	2
NO_RET	3

int declarations and initialization:

mag_select_val = GN_4X_VAL
reticle_selected

ObjectType declarations

round_selected

2.3.6.3.3.1 isu_init

This routine initializes the integrated sight unit (isu).

2.3.6.3.3.2 isu_simul

This routine performs a tick-by-tick simulation of the isu.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp	int	Standard
Errors		
Error Name	**Reason for Error**	
isu_simul: impossible reticle_selected ...	The variable reticle_selected has an unexpected value.	
Calls		
Function	**Where Described**	
controls_hi_mag_reticle	Section 2.3.2	
controls_lo_mag_reticle	Section 2.3.2	
controls_tow_reticle	Section 2.3.2	
controls_no_reticle	Section 2.3.2	
nprintf	Section 2.1.1.3.1.34.1	

Table 2.3-435: isu_simul Information.

2.3.6.3.3.3 isu_gps_mag_12x

This routine sets magnification for gps to 12x.

2.3.6.3.3.4 isu_gps_mag_4x

This routine sets magnification for gps to 4x.

2.3.6.3.3.5 isu_round_select_25mm

This routine selects an M792 round.

2.3.6.3.3.6 isu_round_select_no_round

This routine selects no round.

2.3.6.3.3.7 isu_round_select_tow

This routine selects a tow missile.

2.3.7 Network Interactions

Once each frame, the simulation must process the information coming to it across the SIMNET LAN about other simulated vehicles, weapons effects, collisions, etc. It must also send an update packet on its own status if that has changed significantly in the current frame.

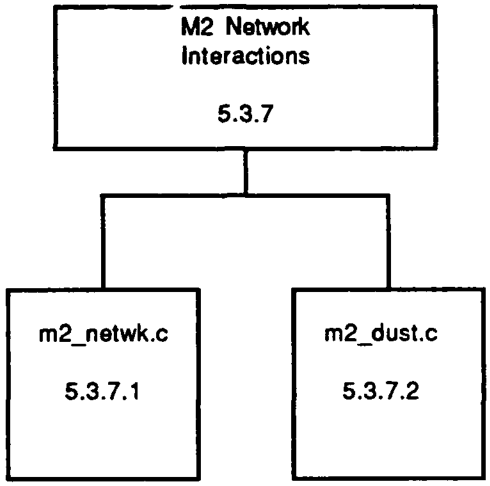

Figure 2.3-8: Structure of the M2 Network Interactions CSC.

The simulation host periodically informs other entities on the network of its current location, orientation, and appearance. If an update in the simulator's appearance is to be sent, libapp calls routines in m2_network.c to fill in vehicle-specific fields. Routines are called in m2_dust.c to determine what size dust cloud, if any, should be reported.

The status of internal subsystems are reported every 30 seconds (for reconstitution and data collection purposes). Routines for sending this status are found in m2_network.c.

The simulation host also sends equipment status packets every 30 seconds, reporting ambient temperature, power supply voltages, and other hardware specific information. The routines for reporting on hardware status are found in the vehicle specific files m2_network.c. This functionality is realized by the following CSU's:

m2_network.c
m2_dust.c

2.3.7.1 m2_network.c
(./simnet/release/src/vehicle/m2/m2_network.c [m2_network.c])

Routines in this CSU are called to update the M2's appearance and the state of its internal systems.

Includes:
"stdio.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libnetwork.h"
"pro_sim.h"
"pro_mgmt.h"
"pro_data.h"
"pro_size.h"
"status_m2.h"
"net/network.h"
"libkin.h"
"libhull.h"
"libfail.h"
"libturret.h"
"libapp.h"
"m2_status.h"
"m2_elecsys.h"
"m2_fuelsys.h"
"m2_ammo.h"
"m2_engine.h"
"m2_odom.h"

2.3.7.1.1 send_equipment_status

This routine sends an equipment status PDU over the network.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
pdu	ManagementPDU	p_data.h
pkt	pointer to register EquipStatusVariant	p_mgmt.h

Calls	
Function	**Where Described**
network_get_simulator_type	Section 2.1.1.3.1.19.1
what_is_voltage12P	Section 2.1.5.2.1
what_is_voltage12N	Section 2.1.5.2.2
what_is_voltage5	Section 2.1.5.2.3
what_is_temperature	Section 2.1.5.2.4
is_host_healthy	Section
is_cig_healthy	Section
is_sound_healthy	Section
is_driver_healthy	Section
is_turret_healthy	Section
network_fill_hdr_send_mgmt_pkt	Section 2.1.1.3.1.42.7
PRO_MGMT_EQUIP_STATUS_SIZE	p_assoc.h

Table 2.3-436: send_equipment_status Information.

2.3.7.1.2 fill_vehicle_spec_status

This routine fills in an M2 vehicle specific status packet.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to register VehicleStatusVariant	p_data.h

Calls	
Function	**Where Described**
engine_get_max_power	Section 2.2.6.2.2.14
electsys_get_hull_battery_voltage	Section 2.3.6.3.1.26
electsys_get_turret_backup_battery_voltage	Section 2.3.6.3.1.27
fuel_level_top	Section 2.3.5.2.10
fuel_level_bottom	Section2.3.5.2.9
ammo_get_apds_can_quantity	Section 2.3.5.1.4
ammo_get_apds_can_ammo_boxes	Section2.3.5.1.5
ammo_get_hei_can_quantity	Section 2.3.5.1.6
ammo_get_hei_can_ammo_boxes	Section2.3.5.1.7
ammo_get_apds_stowed_quantity	Section 2.3.5.1.8
ammo_get_hei_stowed_quantity	Section 2.3.5.1.9
ammo_get_tow_stowed_quantity	Section 2.3.5.1.10
ammo_get_dragon_stowed_quantity	Section 2.3.5.1.11
ammo_get_missile1_val	Section 2.3.5.1.12
ammo_get_missile2_val	Section 2.3.5.1.13
launcher_up_status	Section
ammo_get_m3_configuration_val	Section 2.3.5.1.14
ramp_down_status	Section 2.3.6.1.3.4

Table 2.3-437: fill_vehicle_spec_status Information.

2.3.7.1.3 fill_vehicle_spec_appearance

This routine fills in a vehicle spec appearance packet.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
pkt	pointer to register VehicleAppearanceVariant	p_sim.h
Calls		
Function	**Where Described**	
turret_get_network_azimuth	Section 2.5.5.2.15	
turret_get_network_elevation	Section 2.5.5.2.14	

Table 2.3-438: fill_vehicle_spec_appearance Information.

2.3.7.1.4 network_process_activation_parameters

This routine processes the vehicle's activation parameters, received from the MCC.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
p	pointer to VehicleStatus	status.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
status	pointer to SIMNET_M2_Status	stat_m1.h
Calls		
Function	**Where Described**	
fail_set_subsys	Section 2.5.4.14.2	
sfail_maintenance_condition	Section 2.5.4.26.2	
odom_set_initial_distance_km	Section 2.3.2.3.4.3	
ammo_init_ammo_supply	Section 2.3.5.1.3	
fuel_init_tanks	Section 2.2.5.2.1	
launcher_init_launcher_up	Section 2.3.6.1.4.1	
ramp_init_ramp_down	Section 2.3.6.1.3.1	
engine_init_power	Section 2.3.6.2.6.12	
electsys_init_batteries	Section 2.3.6.3.1.38	

Table 2.3-439: network_process_activation_parameters Information.

2.3.7.1.5 app_init

This routine initializes thresholds.

Errors	
Error Name	**Reason for Error**
Network: couldn't init thresholds	A call to network_init_thresholds returned a zero.
Calls	
Function	**Where Described**
network_init_thresholds	Section 2.1.1.3.1.66.5

Table 2.3-440: app_init Information.

2.3.7.1.6 veh_spec_activate_time

This routine returns 60.

Return Values		
Return Value	**Type**	**Meaning**
60	int	N/A

Table 2.3-441: veh_spec_activate_time Information.

2.3.7.2 m2_dust.c
(./simnet/release/src/vehicle/m2/src/m2_dust.c [m2_dust.c])

The routine in this CSU is called to determine the appearance of the M2's dust cloud.

2.3.7.2.1 tracks_get_dust_cloud

This routine determines the appearance of the M2 Vehicle's trailing dust cloud. The size of the cloud is determined by the vehicle's speed, *speed*, and the *soil_type*.

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
speed	REAL	sim_types.h
soil_type	*int*	*Standard*

Return Values		
Return Value	**Type**	**Meaning**
vehDustCloudNone	int	no dust cloud trails the vehicle
vehDustCloudSmall	int	a small dust cloud trails the vehicle
vehDustCloudMedium	int	a medium dust cloud trails the vehicle
vehDustCloudLarge	int	a large dust cloud trails the vehicle

Errors	
Error	**Reason for Error**
tracks_get_dust_cloud	Invalid soil type

Calls	
Function	**Where Described**
drivetrain_get_vehicle_speed	Section 2.3.6.2.4.8

Table 2.3-442: tracks_get_dust_cloud Information.

2.4 Stealth Vehicle

Figure 2.4-1 depicts the structure of the Stealth Vehicle Software CSC

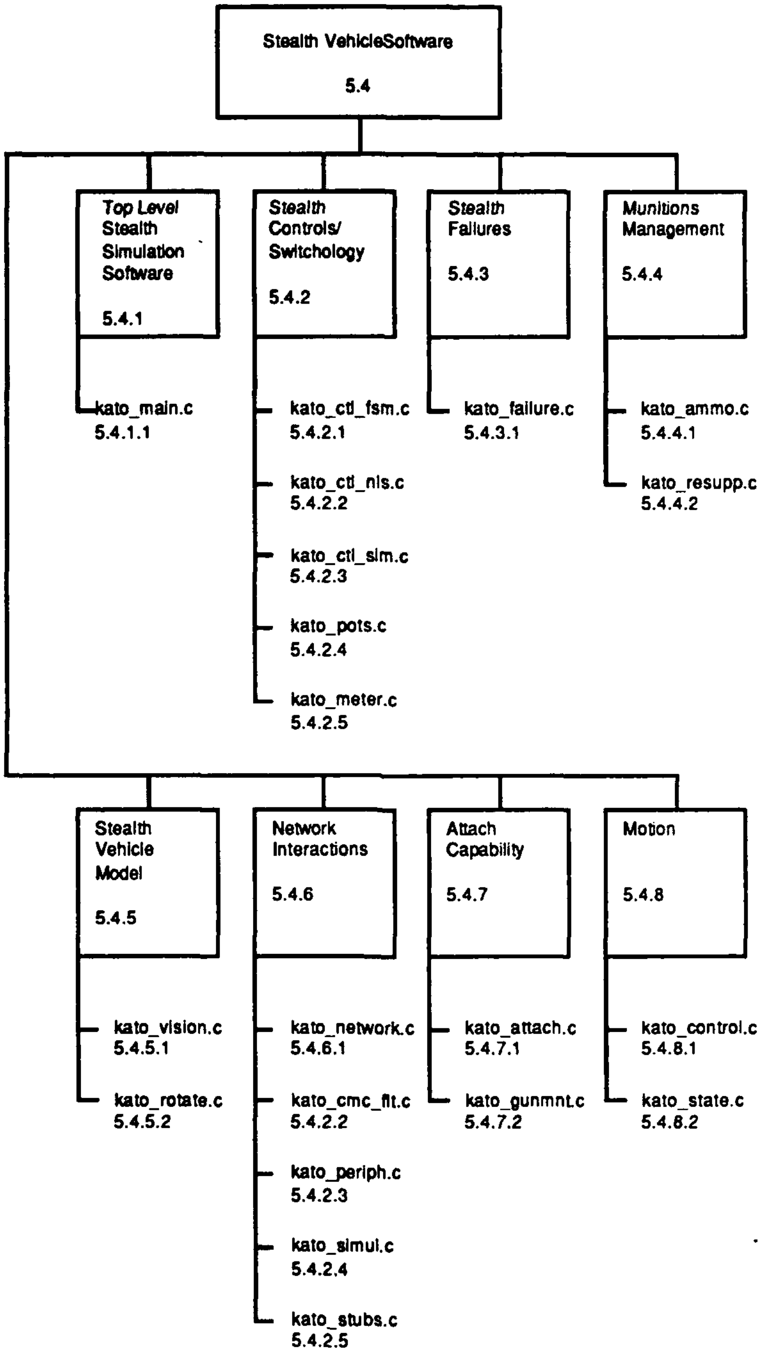

Figure 2.4-1: Structure of the Stealth Vehicle Software CSC.

The CSC'S which constitute this CSC are as follows:

Top Level Stealth Simulation Software
Stealth Controls/Switchology
Failures
Munitions Management
Stealth Vehicle Model
Network Interaction
Attach Capability
Motion

2.4.1 Top Level Stealth Simulation Software

The code required for Stealth simulationis found in this CSC. It contains one CSU, kato_main.c.

2.4.1.1 kato_main.c

(/simnet/release/src/vehicle/kato/src/kato_main.c [kato_main.c])

This file contains routines used in the SIMNET simulation of the Stealth Vehicle.

Includes:

"simstdio.h"	"ctype.h"
"signal.h"	"sim_dfns.h"
"sim_macros.h"	"sim_types.h"
"mass_stdc.h"	"dgi_stdg.h"
"sim_cig_if.h"	"pro_assoc.h"
"pro_sim.h"	"status.h"
"pro_num.h"	"veh_type.h"
"fifo_dfn.h"	"fifo.h"
"bigwheel.h"	"libterrain.h"
"libkin.h"	"libfail.h"
"libcig.h"	"libmsg.h"
"bbd.h"	"libhull.h"
"libidc.h"	"libmain.h"
"libmem.h"	"libnetwork.h"
"librepair.h"	"librva.h"
"libsusp.h"	"libturret.h"
"libsound.h"	"libmap.h"
"timers.h"	"dtad.h"
"status.h"	"ser_status.h"
"cmc.h"	"cmc_timers.h"
"cmc_status.h"	"kato_rnd_dfn.h"
"kato_cntrl.h"	"kato_view.h"
"kato_keybrd.h"	"kato_meter.h"
"kato_pots.h"	"kato_repair.h"
"kato_resupp.h"	"kato_sound.h"
"kato_vision.h"	"kato_status.h"
"kato_tate.h"	

The following are declared:

debug
print_overruns
reboot_on_shutdown
initial_bbd[]
quat_dump()
exit()

The following are declared for the '-p' switch:

init_activ
initial_activation

The following is defined:

PARS_FILE

2.4.1.1.1 print_help

This routine prints out data for the Stealth simulation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
progname	pointer to char	standard

Table 2.4-1: print_help Information.

2.4.1.1.2 print_veh_logo

This routine prints a logo for the Stealth vehicle.

2.4.1.1.3 veh_spec_startup

This routine sets up the network and CIG interfaces at startup.

Calls	
Function	**Where Described**
rtc_init_clock	Section 2.6.16.1.14
AssocSubscribe	Section 2.20.1.1.1
network_get_net_handle	Section 2.1.1.3.2.12.1
network_set_simulator_type	Section 2.1.1.3.1.53.1
network_set_vehicle_class	Section 2.1.1.3.1.54.1
use_cig_reconfig_startup	
cig_set_view_config_file	Section 2.1.2.2.2.23.1
get_vconfig_file1	Section 2.5.1.2.2
map_vehicle_file_read	Section 2.6.11.5.1
get_veh_map_file	Section 2.5.1.2.5
map_read_asid_file	Section 2.6.11.4.1
get_asid_map_file	Section 2.5.1.2.4
map_file_read	Section 2.6.11.3.1
get_ammo_map_file	Section 2.5.1.2.6
keybrd_init	Section 2.1.6.4

Table 2.4-26.4: veh_spec_startup Information.

2.4.1.1.4 veh_spec_idle

This routine is called while the simulator is in the IDLE state.

Calls	
Function	**Where Described**
status_simul	Section 2.1.5.4
keyboard_simul	Section 2.1.6.4
io_simul_idle	Section 2.1.2.2.5.1.2
process_activate_request	Section 2.1.1.3.2.1.1
network_get_exercise_id	Section 2.1.1.3.1.16.1

Table 2.4-3: veh_spec_idle Information.

2.4.1.1.5 veh_spec_init

Order dependent initializations are performed for all of the Stealth's subsystems while the simulator is in the SIMINIT state..

Calls	
Function	**Where Described**
sound_reset	Section 2.1.3.4
status_preset	Section 2.1.5.4
controls_fsm_init	Section 2.4.2.1
controls_sim_init	Section 2.4.2.3
view_init	Section 2.1.2.2.9
meter_init	Section 2.4.2
vision_restore_all_blocks	Section 2.4.5.1
kato_init	Section 2.4.6.4
init_point_to_point	Section 2.4
gunmnt_init	Section 2.4.7.2
controls_edge_init	Section 2.4
app_init	Section
config_pos_init2	Section 2.1.2.2.2.24.2
kinematics_get_o_to_h	Section 2.5.8.2.4
kinematics_get_w_to_h	Section 2.5.8.2.7
resupply_init	Section 2.4.4.2
cig_init_ctr	Section 2.1.2.2.

Table 2.4-5: veh_spec_init Information.

2.4.1.1.6 veh_spec_simulate

This routine calls the routines which simulate the various functins of the Stealth's subsystems on a tick by tick basis.

Calls	
Function	**Where Described**
status_simul	Section 2.1.5.4
keyboard_simul	Section 2.1.6.4
sound_simul	Section 2.1.3.4
controls_simul	Section 2.4
view_simul	Section 2.1.2.2.9
meter_simul	Section 2.4.2.5
kato_simul	Section 2.4
resupply_simul	Section 2.4.4.2

Table 2.4-5: veh_spec_simulate Information.

2.4.1.1.7 veh_spec_stop

This routine is called while the simulator is in the SIMSTOP state. The IDC and sound system hardware are reinitialized, and the vision blocks are broken???

Calls	
Function	**Where Described**
idc_init	Section 2.1.4.1.1.24.1
sound_init	Section 2.1.3.4
vision_break_all_blocks	Section 2.4.5.1

Table 2.4-6: veh_spec_stop Information.

2.4.1.1.8 veh_spec_exit

This routine is called while the simulator is in the SIMEXIT state. Simulation statistics are printed, and the network connection is closed.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
num_ticks	int	standard

Calls	
Function	**Where Described**
attach_exit_gracefully	Section 2.4.7.1
keyboard_exit_gracefully	Section 2.1.6.4
meter_exit_gracefully	Section 2.4.2.5
timers_get_current_time	Section 2.6.3.2.1
timers_get_current_tick	Section 2.6.3.1.1
timers_elapsed_milliseconds	Section 2.6.3.10.1
network_print_statistics	Section 2.1.1.3.2.16.1
net_close	Section 2.20.2.3.1 in MCC CSCI SDD

Table 2.4-7: veh_spec_exit Information.

2.4.1.1.9 main

This routine loops through the simulation of the Stealth vehicle once each frame.The generic simulation routines in "libmain" are called by this routine.

The parameters are parsed:

case -a	Request receive size and request send size are set, and assymetric is set on.
case -b	Bumper numbers are used.
case -B	Spaceball tty is changed.
case -c	Smoothing is shut off.
case -d	Debugging is enabled.
case -e	The ethernet is closed.
case -E	The exercise ID is set.
case -F	Not used
case -f	Not used.
case -g	CIG isn't using graphics.
case -h	Print help.
case -?	Print help.
case -k	The keyboard is used.
case -l	No gunner's magnification is available for Stealth3.
case -n	Nlos missile vehicle mode is enabled.
case -o	Printing of overruns is enabled.
case -p	The simulator is started in stand alone mode. The simulator acts as if it has received an activation packet from the MCC. This segment of code is similar to that used by the MCC for activationg a simulator.
case -r	Restricted flight mode is enabled.
case -S	The network device is set.
case -s	Sound is disabled.
case -t	The named database override is used.
case -T	DED name is set.
case -v	Terrain verbose mode is enabled.
case -u	Vehicle freeze is disabled.
case -1	The CIG1 mask and device are set.
case -2	The CIG2 mask and device are set.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
argc	int	standard
argv	pointer to char	standard

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	standard
initial_heading	float	standard

Calls	
Function	**Where Described**
enter_gracefully	Section 2.6.1.1.1
network_set_exercise_id	Section 2.1.1.3.1.49.1
network_set_net_layer	Section 2.4.6.1
main_read_pars_file	Section 2.5.1.2.1
set_request_receive_size	Section 2.1.2.2.1.29.1
set_request_send_size	Section 2.1.2.2.1.30.1
set_assymetric_on	Section 2.1.2.2.2.112.1
map_set_bumper_status	Section 2.6.11.4.8
controls_set_spaceball_tty	Section 2.4
attach_no_smoothing	Section 2.4.7.1
network_dont_really_open_up_ethernet	Section 2.1.1.3.2.14.1
cig_not_using_graphics	Section 2.1.2.2.1.5.17
print_help	Section 2.4.1.1.1
keyboard_really_use	Section 2.1.6.4
cig_type_lowres	Section 2.1.2.2.8
state_toggle_missile_state	Section 2.4.8.2
get_default_db_name	Section 2.5.1.2.15
get_default_db_version	Section 2.5.1.2.16
state_saf_mode_on	Section 2.4.8.2
network_set_network_device	Section 2.1.1.3.2.12.4
sound_dont_use	Section 2.1.3.4
cig_use_database_0verride_named	Section 2.1.2.2.1.16.1
isalpha	
set_ded_name	Section 2.1.2.2.1.8.2
terrain_verbose_mode_on	Section 2.5.11.9.1
vehicle_freeze_disable	
set_cig_dev	Section 2.1.2.2.1.26.1
set_cig_mask	Section 2.1.2.2.2.114.1
sim_state_startup	Section 2.3.1.1.5
simulation_state_machine	Section 2.5.1.1.13

Table 2.4-8: main Information.

2.4.1.1.10 reconstitute_vehicle

This routine reactivates the Stealth.

Calls	
Function	**Where Described**
vehicle_restart	Section 2.3.19.1.7

Table 2.4-9: reconstitute_vehicle Information.

2.4.2 Stealth Controls/Switchology

Low level controls handling is accomplished with this CSC, which consists of the following CSU's.

kato_ctl_fsm.c
kato_ctl_nls.c
kato_ctl_sim.c

2.4.2.1 kato_ctl_fsm.c
(/simnet/release/src/vehicle/kato/src/kato_ctl_fsm.c [kato_ctl_fsm.c])

This CSU provides the finite power state controls interface for the Stealth Vehicle.

Includes:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"libidc_dfn.h"
"libmem.h"
"libmem_dfn.h"
"libnetwork.h"
"kato_cntrl.h"
"kato_ctl_df.h"
"kato_hard.h" if USE_SPACEBALL not defined
"kato_soft.h"

int declarations:
controls_status
controls_failure_val
controls_failure_edge

Procedure declarations:
controls_sim_next_state()
controls_lamp_init()

2.4.2.1.1 controls_fsm_init

This routine initializes controls_status, controls_failure_val, and controls_failure_edge.

2.4.2.1.2 controls_simul

This routine is called during the Simutation state.

Errors	
Error Name	**Reason for Error**
CONTROLS: controls_simul: Impossible control state	The variable controls_status has an unexpected value.
Calls	
Function	**Where Described**
controls_sim_next_state	Section 2.4.2.1.10
nprintf	Section 2.1.1.3.1.34.1

Table 2.4-*10: controls_simul Information.

2.4.2.1.3 controls_power_status

This routine returns controls status.

Return Values		
Return Value	**Type**	**Meaning**
controls_status	int	status of the controls

Table 2.4-11: controls_power_status Information.

2.4.2.1.4 controls_break_controls

This routine causes the controls to fail.

Errors	
Error Name	**Reason for Error**
DAMAGE: controls broken	This routine was called with controls_failure_val == OFF.
Calls	
Function	**Where Described**
nprintf	Section 2.1.1.3.1.34.1

Table 2.4-12: controls_break_controls Information.

2.4.2.1.5 controls_restore_controls

This routine restores controls.

Calls	
Function	**Where Described**
controls_lamp_init	Section 2.4.2.1.9
controls_fsm_init	Section 2.4.2.1.1
controls_sim_init	Section 2.4.2.2
controls_edge_init	Section 2.4.2.1.8

Table 2.4-13: controls_restore_controls Information.

2.4.2.1.6 controls_failure_status

This routine returns the controls failure status.

Return Values		
Return Value	**Type**	**Meaning**
controls_failure_val	int	failure status

Table 2.4-14: controls_failure_status Information.

2.4.2.1.7 controls_edges_clear

This routine clears edges.

2.4.2.1.8 controls_edge_init

This routine calls controls_edges_clear to initialize edges.

Calls	
Function	**Where Described**
controls_edges_clear	Section 2.4.2.1.7

Table 2.4-15: controls_edge_init Information.

2.4.2.1.9 controls_lamp_init

This routine calls controls_view_ind_init only if USE_SPACEBALL is not defined.

Calls	
Function	**Where Described**
controls_view_ind_init	Section 2.4

Table 2.4-16: controls_lamp_init Information.

2.4.2.1.10 controls_sim_next_state

Calls	
Function	**Where Described**
controls_sim_routines	Section 2.4
controls_sim_off	Section 2.4

Table 2.4-17: controls_sim_next_state Information.

2.4.2.2 kato_ctl_nls.c
(/simnet/release/src/vehicle/kato/src/kato_ctl_nls.c [kato_ctl_nls.c])

This file is not implemented in Version 6.6 of the software.

2.4.2.3 kato_ctl_sim.c
(/simnet/release/src/vehicle/kato/src/kato_ctl_sim.c [kato_ctl_sim.c])

The Stealth's controls interface is provided in part by this CSU.

2.4.2.4 kato_pots.c
(/simnet/release/src/vehicle/kato/src/kato_pots.c [kato_pots.c])

This CSU provides modelling for Stealth's potentiometers. Routines in this file translate hex potentiometer values between 00 and FF into real values and call the appropriate subsystems with these values.

2.4.2.5 kato_meter.c
(/simnet/release/src/vehicle/kato/src/kato_meter.c [kato_meter.c])

This CSU models the Stealth's meters. The appropriate values are set by routines in this file.

2.4.3 Stealth Failures

Stealth vehicle failures are modelled by routines in this CSC. These routines are found in a single CSU, kato_failure.c

2.4.3.1 kato_failure.c

(/simnet/release/src/vehicle/kato/src/kato_failure.c [kato_failure.c])

This file is a stub in anticipation for future use.

2.4.4 Munitions Management

Two CSU's comprise this CSC, kato_ammo.c and kato_resupp.c.

2.4.4.1 kato_ammo.c

(/simnet/release/src/vehicle/kato/src/kato_ammo.c [kato_ammo.c])

This file is not implemented in Version 6.6 of the software.

2.4.4.2 kato_resupp.c

(/simnet/release/src/vehicle/kato/src/kato_resupp.c [kato_resupp.c])

This file is not implemented in Version 6.6 of the software.

2.4.5 Stealth Vehicle Model

The Stealth's subsystems are modelled by routines in this CSC. CSU's required are as follows.

kato_vision.c
kato_rotate.c

2.4.5.1 kato_vision.c

(/simnet/release/src/vehicle/kato/src/kato_vision.c [kato_vision.c])

The routines which control the state of the viewports are found in kato_vision.c. The routines break the viewports (turn them off) and fix them by toggling bits in libvflags.

2.4.5.2 kato_rotate.c

(/simnet/release/src/vehicle/kato/src/kato_rotate.c [kato_rotate.c])

This CSU provides routines which call librotate. They provide the rotation functions for the Stealth Vehicle.

2.4.6 Network Interactions

The Stealth Vehicle has a unique relationship with the Plan View Display System on the SIMNET network. Commands to move the position of the Stealth on the terrain, attach the Stealth to a specific vehicle, or to change the Stealth's exercise ID are all available through the use of the Stealth Protocol.

The CSU's that make up this CSC are as follows:

kato_network.c
kato_cmc_flt.c
kato_periph.c
kato_simul.c
kato_stubs.c

2.4.6.1 kato_network.c
(/simnet/release/src/vehicle/kato/src/kato_network.c [kato_network.c])

When a Stealth Protocol packet is received from the network, the correct routines are called from kato_network.c.

2.4.6.2 kato_cmc_flt.c
(/simnet/release/src/vehicle/kato/src/kato_cmc_flt.c [kato_cmc_flt.c])

This file contains routines which handle the cmc card.

2.4.6.3 kato_periph.c
(/simnet/release/src/vehicle/kato/src/kato_periph.c [kato_periph.c])

Routines in this CSU set up the network and card to look for data packets for the PVD and the p2p protocol.

2.4.6.4 kato_simul.c
(/simnet/release/src/vehicle/kato/src/kato_simul.c [kato_simul.c])

This CSU calls Stealth specific simulation routines and initializes them.

2.4.6.5 kato_stubs.c
(/simnet/release/src/vehicle/kato/src/kato_stubs.c [kato_stubs.c])

This CSU contains a list of functions which are stubbed out for the Stealth. It is necessary for compilation purposes, but provides no functionality.

2.4.7 Attach Capability

The Stealth Vehicle is capable of attaching to another vehicle by searching the list of local vehicles for the one either selected by the PVD, or indicated by a trigger press. The functions in these files search the vehicle list, check for the closest vehicle to the current line of sight, and keep track of the vehicle ID of the attached vehicle. This functionality is realized by the following two CSU's, kato_attach.c and kato_gunmnt.c.

2.4.7.1 kato_attach.c

(/simnet/release/src/vehicle/kato/src/kato_attach.c [kato_attach.c])

2.4.7.2 kato_gunmnt.c

(/simnet/release/src/vehicle/kato/src/kato_gunmnt.c [kato_gunmnt.c])

2.4.8 Motion

The Stealth Vehicle has two independent flight modes (Free Fly, and Terrain Hug) and four attached modes (Tether, Orbit, Compass, and Mimic). Basic inputs from the Spaceball controller are used, in conjunction with position and rotation information from the attached vehicle, to synthesize the Stealth Vehicle's new position and orientation. This functionality is realized by the following CSU's: kato_control.c and kato_state.c.

2.4.8.1 kato_control.c

(/simnet/release/src/vehicle/kato/src/kato_control.c [kato_control.c])

The algorithms which control the motion of the Stealth Vehicle in each of those modes are in this CSU.

2.4.8.2 kato_state.c

(/simnet/release/src/vehicle/kato/src/kato_state.c [kato_state.c])

Information regarding the current state, and transitions into other states are maintained in this CSU.

2.5 Vehicle Libraries

The following figure shows the structure of the Vehicle Libraries CSC. These libraries are used in the vehicle simulation. Functions provided include the following: ballistics modelling, missile modelling, failures and repairs simulation, network interactions, as well as the main simulation steps.

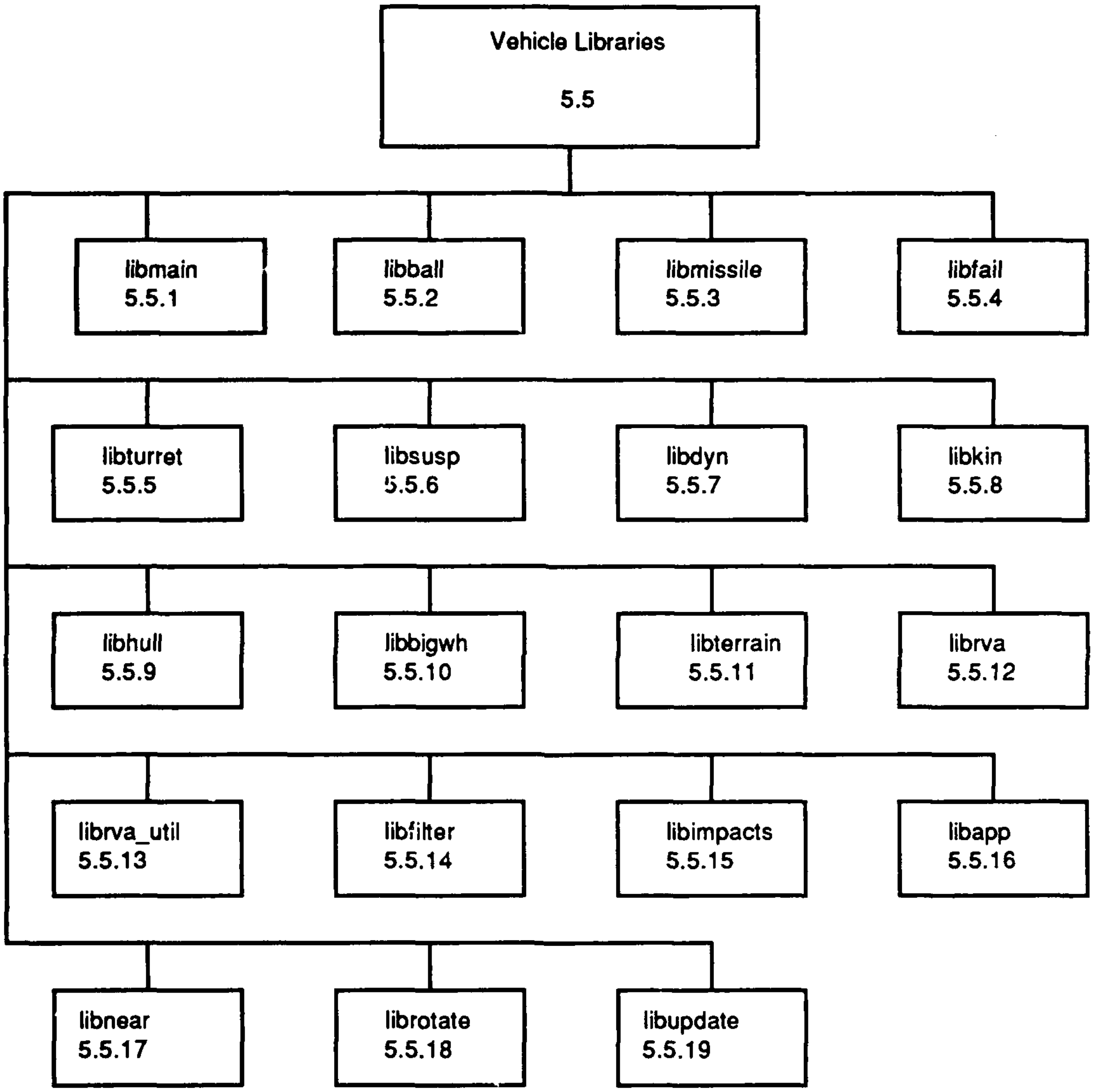

Figure 2.5-1: Vehicle Libraries CSC.

The following CSU's are components of this CSC:

libmain
libball
libmissile
libfail
libturret
libsusp
libdyn
libhull
libkin
libbigwheel
libterrain
librva
librva_util
libfilter
libimpacts
libapp
libnear
librotate
libupdate

2.5.1 libmain
(./simnet/release/src/vehicle/libsrc/libmain [libmain])

In the SIMNET vehicle simulations, the main simulation loop is found in libmain. This loop, executed once each frame, invokes generic functions to perform tasks common to the M1, M2, and Stealth simulation.

2.5.1.1 main.c
(./simnet/release/src/vehicle/libsrc/libmain/main.c)

This file contains all of the generic simulation routines including the main simulation loop.

Includes:

"stdio.h"	- if CLOCK is defined
"clock.h"	- if CLOCK is defined
"signal.h"	- Masscomp only
"sys/mpadvise.h"	- Masscomp only
"sim_dfns.h"	
"sim_macros.h"	
"sim_types.h	
"mass_stdc.h"	
"dgi_stdg.h"	
"sim_cig_if.h"	
"rtc.h"	
"fifo_dfn.h"	
"fifo.h"	
"bigwheel.h"	
"libterrain.h"	
"libkin.h"	
"libfail.h"	
"libcig.h"	
:bbd.h"	
"libhull.h"	
"libidc.h"	
"libmem.h"	
"libmain.h"	
"libnetwork.h"	
"librepair.h"	
"librva.h"	
"libsusp.h"	
"libturret.h"	
"libsound.h"	
"libimps.h"	
"timers.h"	
"dtad.h"	
"status.h"	
"ser_status.h"	

The following are declared:

clock_space	- if CLOCK is defined
clock	- if CLOCK is defined
sim_state	
reboot_on_shutdown	
initial_bbd[]	
cig_startup_func	
first_frame	
exit_gracefully()	
exit()	- for non-Butterfly machines

The following simulation states are defined:

SIM_STARTUP_STATE
SIM_IDLE_STATE
SIM_SIMINIT_STATE
SIM_SIMULATE_STATE
SIM_SIMSTOP_STATE
SIM_SIMEXIT_STATE

2.5.1.1.1 enter_gracefully

This routine enters the simulation. The simulator is put into the idle state, and the vehicle logo is printed on the viewport.

Calls	
Function	**Where Described**
sim_state_idle	Section 2.5.1.1.6
print_veh_logo	Section 2.2.1.1.2

Table 2.5-1: enter_gracefully Information.

2.5.1.1.2 exit_gracefully

This routine exits the simulation. The simulator is put into the simexit state, and a deactivate packet is sent.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
reboot	int	Standard
Calls		
Function	**Where Described**	
sim_state_simexit	Section 2.5.1.1.10	
send_deactivate_pkt	Section 2.1.1.3.1.10.1	

Table 2.5-2: exit_gracefully Information.

2.5.1.1.3 activate_simulation

This routine activates the simulation. The simulator is put into the siminit state.

Calls	
Function	**Where Described**
sim_state_siminit	Section 2.5.1.1.7

Table 2.5-3: activate_simulation Information.

2.5.1.1.4 deactivate_simulation

This routine deactivates the simulation. The simulator is put into the simstop state.

Calls	
Function	**Where Described**
sim_state_simstop	Section 2.5.1.1.9

Table 2.5-4: deactivate_simulation Information.

2.5.1.1.5 sim_state_startup

This routine sets the simulation state to SIM_STARTUP_STATE.

2.5.1.1.6 sim_state_idle

This routine sets the simulation state to SIM_IDLE_STATE.

2.5.1.1.7 sim_state_siminit

This routine sets the simulation state to SIM_SIMINIT_STATE.

2.5.1.1.8 sim_state_simulate

This routine sets the simulation state to SIM_SIMULATE_STATE.

2.5.1.1.9 sim_state_simstop

This routine sets the simulation state to SIM_SIMSTOP_STATE.

2.5.1.1.10 sim_state_simexit

This routine sets the simulation state to SIM_SIMEXIT_STATE.

2.5.1.1.11 sim_state_simulating

This routine returns TRUE if the simulator is in the simulate state and returns FALSE otherwise.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	is in simulation state
FALSE	int	is not in simulation state

Table 2.5-5: sim_state_simulating Information.

2.5.1.1.12 sim_state_sounds_denied

This routine returns FALSE if the simulator is in simulate state and the sound system is disabled, and returns TRUE otherwise.

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	is not in simulation state
FALSE	int	is in simulation state - turn sounds off

Table 2.5-6: sim_state_sounds_denied Information.

2.5.1.1.13 simulation_state_machine

This is the primary routine in this module. It is called by the primary simulation routines in vehicle specific code. For each simulation state, a given set of tasks is performed.

SIM_STARTUP_STATE:

This state is entered once at startup. The simulator hardware and the hull are initialized. Vehicle specific startup routines are called, and the simulator is put into SIM_IDLE_STATE.

SIM_IDLE_STATE:

The simulator is waiting for activation. Timers are started, and vehicle specific routines are called.

SIM_SIMINIT_STATE:

This state is entered once per activation. The simulation is activated, and the CIG is set up. Turret, impacts, repairs, and RVA are initialized, and the simulator is put into SIM_SIMULATE_STATE.

SIM_SIMULATE_STATE:

The simulation is carried out once per tick. Vehicle specific simulation routines as well as failures, kinematics, turret, repairs, network, and I/O simulation routines are called.

SIM_SIMSTOP_STATE:

This state is entered once per deactivation. Vehicle specific stop routines are called. Hull and repairs are uninitialized, the CIG is stopped, and the simulaor is put into SIM_IDLE_STATE.

SIM_SIMEXIT_STATE:

This state is entered once as the simulation is exited. Hardware is uninitialized and shared memory is freed. Vehicle specific routines are called.

Calls	
Function	**Where Described**
bbd_init	Section 2.1.5.1.11.1
dtad_init	Section 2.1.4.2.1.10.2
mem_assign_shared_memory	Section 2.6.12.2.1
ser_heartbeat_init	Section 2.6.7.2.2
idc_init	Section 2.1.4.1.1.24.1
sound_init	Section 2.1.3.2.4
status_init	Sections 2.1.5.2 m1_status.c, 2.1.5.3 m2_status.c, and 2.1.5.4 kato_status.c
timers_init	Section 2.6.3.8.1
pots_init	Sections 2.2.2.3.2 m1_pots.c, 2.3.2.1.4 m2_pots.c, and 2.4.2.4 kato_pots.c
cig_prepare	Section 2.1.2.2.1.6.1
buffer_setup	Section 2.1.2.2.2.16.1
cig_synchronize	Section 2.1.2.2.1.13.1
repair_uninit	Section 2.5.4.19.2
hull_init	Section 2.5.9.1.1
cig_stop	Section 2.1.2.2.1.12.1
network_init	Section 2.1.1.3.2.12.3
network_can_i_really_use_network	Section 2.1.1.3.2.27.1
filter_init	Section 2.5.14.7.1
network_use_network_handles	Sections 2.2.7.1 m1_network.c, 2.3.7.1 m2_network.c, and 2.4.6.1 kato_network.c
rva_setup	Section 2.5.12.25.1
get_priority_list_file	Section 2.5.1.2.10
sim_state_idle	Section 2.5.1.1.6
veh_spec_startup	Section 2.2.1.1.3, 2.3.1.1.3, and 2.4.1.1.3
timers_simul	Section 2.6.3.15.1
veh_spec_idle	Section 2.2.1.1.4, 2.3.1.1.4, and 2.4.1.1.4
init_ballistics_buffer	Section 2.1.2.2.2.14.1
idc_reset	Section 2.1.4.1.1.24.4
veh_spec_init	Section 2.2.1.1.5, 2.3.1.1.5, and 2.4.1.1.5
impacts_init	Section 2.5.15.1.1
turret_init	Sections 2.2.6.1.1 m1_turret.c and 2.3.6.1.1 m2_turret
repair_init	Sections 2.2.4.2.3 and 2.3.4.2.3
timers_init_starttime	Section 2.6.3.16.1
rva_init	Section 2.5.12.22.1
buffer_reset	Section 2.1.2.2.2.15.1
cig_spec_init	Section 2.1.2.2.6.7
fail_init	Section 2.2.4.1.1 and 2.3.4.1.1
sim_state_simulate	Section 2.5.1.1.8

rtc_start_time	Section 2.6.16.1.2
bbd_bit_out	Section 2.1.5.1.4.1
timers_simul	Section 2.6.3.15.1
rtc_stop_time	Section 2.6.16.1.3
fail_simul	Section 2.5.4.13.1
veh_spec_simulate	Section 2.2.1.1.6, 2.3.1.1.6, and 2.4.1.1.6
kinematics_simul	Section 2.5.8.6.1
turret_simul	Sections 2.2.6.1.1 m1_turret.c and 2.3.6.1.1 m2_turret.c
repair_simul	Sections 2.2.4.2.2 amd 2.3.4.2.2
net_simul	Sections 2.2.7.1 m1_network.c, 2.3.7.1 m2_network.c, and 2.4.6.1 kato_network.c
io_simul	Section 2.1.2.2.5.1.1
veh_spec_stop	Section 2.2.1.1.7, 2.3.1.1.7, and 2.4.1.1.7
hull_uninit	Section 2.5.9.1.2
sound_reset	Section 2.1.3.2.7
cig_uninit	Section 2.1.2.2.1.14.1
dtad_uninit	Section 2.1.4.2.1.11.1
bbd_uninit	Section 2.1.5.1.16.1
veh_spec_exit	Section 2.2.1.1.8
mem_free_shared_memory	Section 2.6.12.1.2

Table 2.5-7: simulation_state_machine Information.

2.5.1.2 read_pars.c
(./simnet/release/src/vehicle/libsrc/libmain/read_pars.c)

This file contains routines for reading parameter files. It allows one to specifiy vehicle specific file names in a parameter file, such as the default database name, the default ded name, and file names for the mapping files.

Includes:
"stdio.h"
"simstdio.h"
basic.h"

The following are declared:
alternate_pars_file[80]
vconfig_file1[80]
vconfig_file2[80]
asid_map_file[80]
veh_map_file[80]
ammo_map_file[80]
sdamage_file[80]
thresh_file[80]
idle_filter_file[80]
sim_filter_file[80]
priority_list_file[80]
register_file[80]
devices_file[80]
calib_file[80]
default_db_name[TerrainNameLength]
default_db_version
ded_override[32]
db_override[32]
constatus_file[80]

2.5.1.2.1 main_read_pars_file

This routine reads the parameter file specified by *fn*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
fn	pointer to char	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
strtok()	pointer to char	Standard
fp	pointer to FILE	Standard
s[80]	char	Standard
str	pointer to char	Standard
Calls		
Function	**Where Described**	
main_read_pars_file	Section 2.5.1.2.1	
cig_use_database_override_named	Section 2.1.2.2.1.16.1	
set_ded_name	Section 2.1.2.2.1.8.2	

Table 2.5-8: main_read_pars_file Information.

2.5.1.2.2 get_vconfig_file1

This routine returns a pointer to a character string, *vconfig_file1,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
vconfig_file1	pointer to char	represents vconfig file1 file name

Table 2.5-9: get_vconfig_file1 Information.

2.5.1.2.3 get_vconfig_file2

This routine returns a pointer to a character string, *vconfig_file2,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
vconfig_file2	pointer to char	represents vconfig file2 file name

Table 2.5-10: get_vconfig_file2 Information.

2.5.1.2.4 get_asid_map_file

This routine returns a pointer to a character string, *asid_map_file,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
asid_map_file	pointer to char	represents asid map file name

Table 2.5-11: get_asid_map_file Information.

2.5.1.2.5 get_veh_map_file

This routine returns a pointer to a character string, *veh_map_file,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
veh_map_file	pointer to char	represents vehicle mapping file name

Table 2.5-12: get_veh_map_file Information.

2.5.1.2.6 get_ammo_map_file

This routine returns a pointer to a character string, *ammo_map_file,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
ammo_map_file	pointer to char	represents ammo map file name

Table 2.5-13: get_ammo_map_file Information.

2.5.1.2.7 get_sdamage_file

This routine returns a pointer to a character string, *sdamage_file,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
sdamage_file	pointer to char	represents sdamage file name

Table 2.5-14: get_sdamage_file Information.

2.5.1.2.8 get_thresh_file

This routine returns a pointer to a character string, *thresh_file,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
thresh_file	pointer to char	represents thresh file name

Table 2.5-15: get_thresh_file Information.

2.5.1.2.9 get_idle_filter_file

This routine returns a pointer to a character string, *idle_filter_file,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
idle_filter_file	pointer to char	irepresents dle filter file name

Table 2.5-16: get_idle_filter_file Information.

2.5.1.2.10 get_priority_list_file

This routine returns a pointer to a character string, *priority_list_file,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
priority_list	pointer to char	represents priority list file name

Table 2.5-17: get_priority_list_file Information.

2.5.1.2.11 get_register_file

This routine returns a pointer to a character string, *register_file,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
register_file	pointer to char	pointer to register file name

Table 2.5-18: get_register_file Information.

2.5.1.2.12 get_device_file

This routine returns a pointer to a character string, *device_file,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
device_file	pointer to char	represents device file name

Table 2.5-19: get_device_file Information.

2.5.1.2.13 get_calib_file

This routine returns a pointer to a character string, *calib_file,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
calib_file	pointer to char	represents calibration file name

Table 2.5-20: get_calib_file Information.

2.5.1.2.14 get_sim_filter_file

This routine returns a pointer to a character string, *sim_filter_file,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
sim_filter_file	pointer to char	represents sim filter file name

Table 2.5-21: get_sim_filter_file Information.

2.5.1.2.15 get_default_db_name

This routine returns a pointer to a character string, *default_db_name,* which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
default_db_name	pointer to char	represents default database name

Table 2.5-22: get_default_db_name Information.

2.5.1.2.16 get_default_db_version

This routine returns *default_db_version*..

Return Values		
Return Value	**Type**	**Meaning**
default_db_version	unsigned short	default database version

Table 2.5-23: get_default_db_version Information.

2.5.1.2.17 get_ded_override

This routine returns a pointer to a character string, *ded_override*.

Return Values		
Return Value	**Type**	**Meaning**
ded_override	pointer to char	ded override

Table 2.5-24: get_ded_override Information.

2.5.1.2.18 get_db_override

This routine returns *db_override*.

Return Values		
Return Value	**Type**	**Meaning**
db_override	pointer to char	data base override

Table 2.5-25: get_db_override Information.

2.5.1.2.19 get_constants_file

This routine returns a pointer to a character string, *constants_file*, which represents the file name.

Return Values		
Return Value	**Type**	**Meaning**
constants_file	pointer to char	represents constants file name

Table 2.5-26: get_constants_file Information.

2.5.1.2.20 print_pars_files

This routine prints each of the parameter files.

Calls	
Function	**Where Described**
get_vconfig_file1	Section 2.5.1.2.2
get_vconfig_file2	Section 2.5.1.2.3
get_asid_map_file	Section 2.5.1.2.4
get_veh_map_file	Section 2.5.1.2.5
get_ammo_map_file	Section 2.5.1.2.6
get_sdamage_file	Section 2.5.1.2.7
get_thresh_file	Section 2.5.1.2.8
get_idle_filter_file	Section 2.5.1.2.9
get_sim_filter_file	Section 2.5.1.2.14
get_priority_list_file	Section 2.5.1.2.10
get_register_file	Section 2.5.1.2.11
get_default_db_name	Section 2.5.1.2.15
get_default_db_version	Section 2.5.1.2.16
get_ded_override	Section 2.5.1.2.17
get_db_override	Section 2.5.1.2.18
get_constants_file	Section 2.5.1.2.19

Table 2.5-27: print_pars_files Information.

2.5.2 libball

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

Libball is used to convert data from the given firing tables containing range, time of flight, and superelevation data into ballistics tables containing range, distance away, and distance down on a 30 Hz cycle. This ballistics table is sent to the CIG at initialization.

2.5.2.1 ball_calc.c

(./simnet/release/src/vehicle/libsrc/libball/ball_calc.c)

Inlcudes:

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

Defines:

MAX_NUMBER_ITERATIONS
CONVERGENCE_CRITERION

2.5.2.1.1 ballistics_calc_time

This routine computes and returns the time required for a certain type of ammunition to fly the specified range. The routine uses the Newton-Raphson method of iteration to solve for time given the range and the function R = f(t). On call, the function is passed pointers to arrays containing the polynomial coefficients for the functions xb = f(t), yb = f(t), and the value of the range desired.

Parameters are represented as follows:
xb_coefficients -- polynomial coefficient
yb_coefficients -- polynomial coefficient
desired_range -- desired range

Parameters		
Parameter	**Type**	**Where Typedef Declared**
*xb_coefficients	pointer to REAL	/simnet/common/include/global/sim_types.h
*yb_coefficients	pointer to REAL	/simnet/common/include/global/sim_types.h
desired_range	REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
time	register REAL	/simnet/common/include/global/sim_types.h
current_xb	register REAL	/simnet/common/include/global/sim_types.h
current_yb	register REAL	/simnet/common/include/global/sim_types.h
current_xb_prime	register REAL	/simnet/common/include/global/sim_types.h
current_yb_prime	register REAL	/simnet/common/include/global/sim_types.h
current_range	REAL	/simnet/common/include/global/sim_types.h
current_range_prime	REAL	/simnet/common/include/global/sim_types.h
error	REAL	/simnet/common/include/global/sim_types.h
i	int	standard
Return Values		
Return Value	**Type**	**Meaning**
time	REAL	time required to fly the specified range
1.0	REAL	procedure failed

Errors	
Error	**Reason for Error**
stderr	Ballistics - no convergence on flight time

Table 2.5-28: ballistics_calc_time Information.

2.5.2.1.2 ballistics_calc_se

This routine computes and returns the superelevation required for certain type of ammunition to fly the specified range. On call, the function is passed pointers to arrays containing the polynomial coefficients for the functions xb = f(t), yb = f(t), and the value of the range desired. The superelevation angle is returned in radians.

Parameters are represented as follows:
xb_coefficients -- polynomial coefficient
yb_coefficients -- polynomial coefficient
desired_range -- desired range

Parameters		
Parameter	**Type**	**Where Typedef Declared**
*xb_coefficients	pointer to REAL	/simnet/common/include/global/sim_types.h
*yb_coefficients	pointer to REAL	/simnet/common/include/global/sim_types.h
range	REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
superelevation	REAL	/simnet/common/include/global/sim_types.h
tof	REAL	/simnet/common/include/global/sim_types.h
xb	REAL	/simnet/common/include/global/sim_types.h
yb	REAL	/simnet/common/include/global/sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
superelevation	REAL	the superelevation angle required to fly the specified range
Calls		
Function	**Where Described**	
ballistics_calc_time	Section 2.5.2.1.1	

Table 2.5-29: ballistics_calc_se Information.

2.5.2.2 ball_fire.c
(./simnet/release/src/vehicle/libsrc/libball/ball_fire.c)

Includes:
"stdio.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"libmatrix.h"
"libmsg.h"
"libevent.h"

2.5.2.2.1 ballistics_fire_a_round

This routine sends ballistics data to the CIG system when firing a round.

Parameters are represented as follows:

ammo -- the ammunition type
gun_position -- the gun's location in world coordinates
gun_velocity -- the gun's volocity in world coordinates
gun_to_world -- the transfer matrix for gun to world coordinates
tracer_lit -- the flag indicating if any tracers
round_id -- the event flag for the round (sent to the CIG, then received back upon impact)

Parameters		
Parameter	**Type**	**Where Typedef Declared**
ammo	int	standard
gun_position	VECTOR	/simnet/common/include/global/sim_types.h
gun_velocity	VECTOR	/simnet/common/include/global/sim_types.h
gun_to_world	T_MATRIX	/simnet/common/include/global/sim_types.h
tracer_lit	int	standard
round_id	int	standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
xy_proj	REAL	/simnet/common/include/global/sim_types.h
az_sin	REAL	/simnet/common/include/global/sim_types.h
az_cos	REAL	/simnet/common/include/global/sim_types.h

Calls	
Function	**Where Described**
store_round_fired	Section 2.1.2.2.2.14.4

Table 2.5-30: ballistics_fire_a_round Information.

2.5.2.3 ball_load.c

(./simnet/release/src/vehicle/libsrc/libball/ball_load.c)

Includes:

"stdio.h"
"math.h"
"sim_dfns.h"
"sim_types.h"
"sim_macros.h"
"basic.h"
"libmap.h"
"libmsg.h"

Defines:

MAX_BALLISTIC_TABLE_SIZE

2.5.2.3.1 ballistics_load_trajectory_file

This routine is passed a ballistics data (.d) file containing the trajectory points. It downloads these points to the CIG.

Parameters are represented as follows:
file -- the name of the ballistics data file to be downloaded to the CIG
ammo_type -- the ammunition type

Parameters		
Parameter	**Type**	**Where Typedef Declared**
*file	pointer to char	standard
ammo_type	ObjectType	/simnet/common/include/protocol/p_sim.h
ammo_index	int	standard

Internal Variables		
Variable	**Type**	**Where Typedef Declared**
*fp	pointer to FILE	
num_entries	int	standard
i	int	standard
time[MAX_BALLISTIC_TABLE_SIZE]	REAL	/simnet/common/include/global/sim_types.h
yb[MAX_BALLISTIC_TABLE_SIZE]	REAL	/simnet/common/include/global/sim_types.h
zb[MAX_BALLISTIC_TABLE_SIZE]	REAL	/simnet/common/include/global/sim_types.h
ammo	int	standard

Errors	
Error	**Reason for Error**
stderr	Cannot open file

Calls	
Function	**Where Described**
map_get_ammo_entry_from_network_type	Section 2.6.11.2.1
cig_msg_append_traj_table_xfer	Section 2.1.2.2.2.11.1

Table 2.5-31: ballistics_load_trajectory_file Information.

2.5.2.3.3 ballistics_load_parameter_file

This routine downloads a parameter file (.p) into the simulation. The file contains two polynomials: 1) the range vs. time, and 2) the drop vs. time. These polynomial are used to generate the ballistics curve passed to the CIG.

Parameters are represented as follows:
file -- the name of the parameter file to be downloaded
yb_coeff[] -- the distance y component (distance out)
zb_coeff[] -- the distance z component (distance drop)

Parameters		
Parameter	**Type**	**Where Typedef Declared**
*file	pointer to char	standard
yb_coeff[]	REAL	/simnet/common/include/global/sim_types.h
zb_coeff[]	REAL	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
*fp	pointer to FILE	
Errors		
Error	**Reason for Error**	
stderr	Cannot open file	

Table 2.5-32: ballistics_load_parameter_file Information.

2.5.2.4 ball_orient.c
(./simnet/release/src/vehicle/libsrc/libball/ball_orient.c)

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

Variable Declarations:
o_mat

2.5.2.4.1 ballistics_cal_azm_elev

This routine calculates the azimuth and elevation from the direction cosine matrix.

This routine is given a gun to world matrix, *m*. The routine passes a pointer to the matrix which contains:
[1,1] -- the sine of the elevation angle
[1,2] -- the cosine of the elevation angle
[2,1] -- the sine of the azimuth
[2,2] -- the cosine of the azimuth
[3,1] -- zero
[3,2] -- zero

Parameters		
Parameter	**Type**	**Where Typedef Declared**
m	T_MATRIX	/simnet/common/include/global/sim_types.h
Internal Variables		
Variable	**Type**	**Where Typedef Declared**
elevation_sin	REAL	/simnet/common/include/global/sim_types.h
elevation_cos	REAL	/simnet/common/include/global/sim_types.h

Table 2.5-33: ballistics_calc_azm_elev Information.

2.5.3 libmissile
(./simnet/release/src/vehicle/libsrc/libmissile libmissile])

This CSU provides functions for the launching, flying, and detonation of various types of missiles.

2.5.3.1 fuze_prox.c
(./simnet/release/src/vehicle/libsrc/libmissile/fuze_prox.c)

This file contains the code which is called by specific missiles with proximity fuses to determine if the missile should be detonated due to its close proximity to a vehicle.

Includes:
"stdio.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"basic.h"
"/simnet/common/include/protocol/p_sim.h"
"libnear.h"
"libmatrix.h"
"libmiss_dfn.h"
"libmiss_loc.h"

Defines:
NUM_PROX

Declarations:
prox_list [NUM_PROX]
prox_free [NUM_PROX]
free_ptr
malloc()

The following routines are declared static to this module:
get_prox()
free__prox()
dfd_vec_sub()
f2d_vec_scale()
f2d_mat_transpose()

2.5.3.1.1 missile_fuze_prox_init

This routine sets up the statically allocated memory. Each element of *prox_list* is statically allocated and a pointer is assigned to each element in *prox_free*. *free_ptr* is set to point to the first element in *prox_free*.

Internal Variables		
Internal Variable	Type	Where Typedef Declared
i	int	Standard

Table 2.5-34: missile_fuze_prox_init Information.

2.5.3.1.2 missile_fuze_prox

This routine determines which vehicles are close enough to the missile to be considered potential targets of the missile. A linked list of three potential targets is maintained. Given the current position and velocity of both a missile and a target, this routine determines if their proximity is close enough to detonate the missile. If they are close enough to cause a detonation, but a closer approach is predicted, detonation is guaranteed but delayed to allow for the closer detonation. This routine deals with a single time step. During this time step, both the missile and target fly along paths which can be described as chords in space. The chords are defined by initial points and direction vectors whose lengths are equal to the lengths of the chords. These chords are velocity vectors measured in units of distance per tick. Both the target and missile are assumed to travel the lengths of their respective chords at a constant rate. The distance traveled is described by a parameter which varies from -1.0 to 0.0. The parameter for both chords is always the same.

The parameters are defined as follows:

mptr	- a pointer to the missile whose fuse is being processed.
target_flag	- a flag that indicates which set of vehicles are to be tested for fuse detonation. If it is set to PROX_FUZE_ON_NO_VEH , the fuse is not armed. The fuse will only detonate when it is near the specified vehicle if it is set to PROX_FUZE_ON_ONE_VEH. A value of PROX_FUZE_ON_ALL_VEH will allow the fuse to detonate when it is close enough to any vehicle in the vehicle list.
targ_vehicle_id	- a pointer to the vehicle ID of the vehicle which will cause the missile to detonate if it is close enough.
first_targ	- a pointer to the first PROX element on the list of the missile. Various targets are maintained by using a linked list.
veh_list	- the list to check for possible targets.
invest_dist_2	- the square of the distance within which the target must be for the missile to maintain it as a possible detonation.
prox_dist_2	- the square of the distance between the missile and the target which will cause the missile to detonate.

The internal variables are defined as follows:

target	- a pointer to an appearance packet of a target.
veh_count	- an index to the vehicles list.
current_targ	- a pointer to a *PROX* element.
missile_vel	- the velocity vector of the missile.
missile_pos	- a pointer to the missile location vector.
not_found_expl	- a flag which indicates that a target which cannot be found during this tick would have caused a detonation during the last tick, but a closer approach was predicted.
temp_targ	- a temporary pointer to a *PROX* element.
vel_diff	- a vector difference between the missile and target velocities.
term_diff	- the vector difference between the initial points of the missile and target pathway chords.
vel_dot_vel	- the dot product of *vel_diff* with itself.
term_dot_term	- the dot product of *term_diff* with itself.
term_dot_vel	- the dot product of *term_diff* with *vel_diff*.
closest_approach	- the value of the chord position parameter where the closest approach occurs.

min_allowed_approach - the value of the chord position parameter where the closest approach occurs between -1.0 and 0.0.
min_prox_dist_2 - the square of the closest approach distance.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
target_flag	int	Standard
targ_vehicle_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
first_targ	pointer to a pointer to PROX	Section 2.5.3.2
veh_list	int	Standard
invest_dist_2	REAL	/simnet/common/include/protocol/sim_types.h
prox_dist_2	REAL	/simnet/common/include/protocol/sim_types.h

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
target	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
veh_count	int	Standard
current_targ	pointer to register PROX	Section 2.5.3.2
i	register int	Standard
missile_vel	VECTOR	/simnet/common/include/protocol/sim_types.h
missile_pos	pointer to a REAL	/simnet/common/include/protocol/sim_types.h
not_found_expl	int	Standard
temp_targ	pointer to PROX	Section 2.5.3.2
vel_diff	VECTOR	/simnet/common/include/protocol/sim_types.h
term_diff	VECTOR	/simnet/common/include/protocol/sim_types.h
vel_dot_vel	REAL	/simnet/common/include/protocol/sim_types.h
term_dot_term	REAL	/simnet/common/include/protocol/sim_types.h
term_dot_vel	REAL	/simnet/common/include/protocol/sim_types.h
closest_approach	REAL	/simnet/common/include/protocol/sim_types.h
min_allowed_approach	REAL	/simnet/common/include/protocol/sim_types.h
min_prox_dist_2	REAL	/simnet/common/include/protocol/sim_types.h

Calls	
Function	**Where Described**
vec_scale	Section 2.6.2.64
near_get_next_veh_near_point	Section 2.5.17.1.1
get_prox	Section 2.5.3.1.4

VEHICLE_IDS_EQUAL	/simnet/common/include/global/sim_macros.h
near_get_veh_if_still_near_point	Section 2.5.17.1.2
free_prox	Section 2.5.3.1.5
dfd_vec_sub	Section 2.5.3.1.7
vec_sub	Section 2.6.2.65
vec_dot_prod	Section 2.6.2.54
vec_scale	Section 2.6.2.64
vec_add	Section 2.6.2.57
f2d_vec_scale	Section 2.5.3.1.8
f2d_mat_transpose	Section 2.5.3.1.6
missile_util_comm_fuze_detonate	Section 2.5.3.25.6

Table2.5-35: missile_fuze_prox Information.

2.5.3.1.3 missile_fuze_prox_stop

This routine frees up *PROX* memory associated with a particular missile. *first_targ* is a pointer to a pointer to a PROX list. *targ* is a pointer to a *PROX* list.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
first_targ	pointer to pointer to PROX	Section 2.5.3.2
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
targ	pointer to PROX	Section 2.5.3.2
Calls		
Function	**Where Described**	
free_prox	Section 2.5.3.1.5	

Table 2.5-36: missile_fuze_prox_stop Information.

2.5.3.1.4 get_prox

This routine finds a free entry in *prox_list* if one exists. Pointers to free elements in *prox_list* are kept in *prox_free*. If none of the elements in *prox_list* are free, a *PROX* element is allocated from memory and a pointer is returned to it. The routine records if memory has been allocated. prox_alloc is a pointer to a PROX element allocated from memory.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
prox_alloc	pointer to PROX	Section 2.5.3.2
Return Values		
Return Value	**Type**	**Meaning**
prox_free[free_ptr++]	static	a free entry in prox_list
prox_alloc	static	pointer to a PROX element allocated from memory

Table 2.5-37: get_prox Information.

2.5.3.1.5 free_prox

This routine frees the memory of a *PROX* element if it is no longer needed. If it is an element of *prox_list*, a pointer to it is put into *prox_free*; otherwise, the memory is restored to the stack. *prox_ptr* is a pointer to an element to be freed.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
prox_ptr	pointer to PROX	Section 2.5.3.2

Table 2.5-38: free_prox Information.

2.5.3.1.6 f2d_mat_transpose

This routine puts the transpose of the (float) matrix *src* into the (REAL) matrix *dst*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
src	3 by 3 matrix of float	Standard
dst	3 by 3 matrix of REAL	/simnet/common/include/protocol/sim_types.h

Table 2.5-39: f2d_mat_transpose Information.

2.5.3.1.7 dfd_vec_sub

This routine finds *result = v1-v2*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v1	VECTOR	/simnet/common/include/protocol/sim_types.h
v2	array 3 of float	Standard
result	VECTOR	/simnet/common/include/protocol/sim_types.h

Table 2.5-40: dfd_vec_sub Information.

2.5.3.1.8 f2d_vec_scale

This routine scales the float vector *v* by *scale_factor* and puts the result in the (REAL) vector *result*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
v	array 3 of float	Standard
scale_factor	REAL	/simnet/common/include/protocol/sim_types.h
result	VECTOR	/simnet/common/include/protocol/sim_types.h

Table 2.5-41: f2d_vec_scale Information.

2.5.3.2 libmiss_dfn.h
(./simnet/release/src/vehicle/libsrc/libmissile/libmiss_dfn.h)

This file contains type definitions used only with the missile library.

2.5.3.3 libmiss_loc.h
(./simnet/release/src/vehicle/libsrc/libmissile/libmiss_loc.h)

This file contains function declarations used only within the missile library.

2.5.3.4 libmissile.h
(./simnet/release/src/vehicle/libsrc/libmissile/libmissile.h)

This file contains function definitions used both inside and outside the missile library.

2.5.3.5 miss_adat.c

(./simnet/release/src/vehicle/libsrc/libmissile/miss_adat.c)

This file contains routines which fly out a missile with the characteristics of an ADAT missile.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
basic.h"
"mun_type.h"
"libmap.h"
"libmatrix.h"
miss_adat.h"
"libmiss_dfn.h"
"libmiss_loc.h"

The following define the missile characteristics:

ADAT_BURNOUT_TIME
ADAT_MAX_FLIGHT_TIME
INVEST_DIST_SQ
HELO_FUZE_DIST_SQ
AIR_FUZE_DIST_SQ
ADAT_TEMP_BIAS_TIME
CLOSE_RANGE

The following define the possible states of the ADAT_MISSILE:

ADAT_FREE
ADAT_GUIDE
ADAT_UNGUIDE
ADAT_CLOSE
ADAT_NOT

The following terms set the order of the polynomials used to determine the speed or the cosine of the maximum allowed turn rate of the missile at any point in time:

ADAT_BURN_SPEED_DEG
ADAT_COAST_SPEED_DEG
ADAT_BURN_TURN_DEG
ADAT_COAST_TURN_DEG
ADAT_TEMP_BIAS_DEG

Coefficients for the speed polynomial before motor burnout:

adat_burn_speed_coeff [ADAT_BURN_SPEED_DEG + 1]

Coefficients for the speed polynomial after motor burnout:

adat_coast_speed_coeff [ADAT_COAST_SPEED_DEG + 1]

Coefficients for the cosine of maximum turn polynomial before motor burnout:

adat_burn_turn_coeff [ADAT_BURN_TURN_DEG + 1]

Coefficients for the cosine of maximum turn polynomial after motor burnout:
adat_coast_turn_coeff [ADAT_COAST_SPEED_DEG + 1]

Coefficients for the temporal bias polynomial:
adat_temp_bias_coeff [ADAT_TEMP_BIAS_DEG + 1]

The following arrays are used to give the missile the proper superelevation at launch time. Two are required to deal with launches off either side of the turret.
tube_C_sight_left
tube_C_sight_right

Memory for the missiles is declared in vehicle specific code. During initialization, a pointer is assigned to this memory, then some memory issues are handled in this module. *adat_array* is a pointer to missile memory, and *num_adats* is the number of defined missiles.

The following functions are declared as static:
missile_adat_fly()
missile_adat_stop()

2.5.3.2.1 missile_adat_init

This routine copies the parameters into variables static to this module and initializes the state of all the missiles. It also initializes the proximity fuse. *missile_array[]* is a pointer to an array of ADAT missiles in vehicle specific code. *num_missiles* is the number of missiles defined in *missile_array*. *mag* is used to generate tube to sight matrices.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
missile_array	array of ADAT_MISSILE	Section 2.5.3.6
num_missiles	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
mag	REAL	/simnet/common/include/protocol/sim_types.h
Calls		
Function	**Where Described**	
missile_fuze_prox_init	Section 2.5.3.1.1	
mat_copy	Section 2.6.2.39.1	

Table 2.5-42: missile_adat_init Information.

2.5.3.2.2 missile_adat_fire

This routine performs the functions specifically related to the firing of an ADAT missile. The parameters are defined as follows:
aptr - a pointer to the ADAT missile to be fired.

target_type - the type of target for which the missile can be set by the launching vehicle.
launch_point - the location, in world coordinates, from which the missile is launched.
loc_sight_to_world - the sight to world transformation matrix used only in this routine.
launch_speed - the speed of the launch platform (assumed to be in the direction of the missile).
range_to_intercept - the range to intercept.
tube - the tube from which the missile was launched.
target_vehicle_id - the vehicle ID of the target (if any).

The relevant internal variables are as follows:

mptr - the pointer to the particular generic missile to which *aptr* points.
comm_target_type - an indication of whether or not the target is known.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
aptr	pointer to ADAT_MISSILE	Section 2.5.3.6
target_type	int	Standard
launch_point	VECTOR	/simnet/common/include/protocol/sim_types.h
loc_sight_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h
launch_speed	REAL	/simnet/common/include/protocol/sim_types.h
range_to_intercept	REAL	/simnet/common/include/protocol/sim_types.h
tube	int	Standard
target_vehicle_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
mptr	pointer to MISSILE	Section 2.5.3.2
comm_target_type	int	Standard

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	successful
FALSE	int	not successful

Calls	
Function	**Where Described**
vec_copy	Section 2.6.2.59
mat_copy	Section 2.6.2.39.1
mat_mat_mul	Section 2.6.2.32.1
missile_util_comm_fire_missile	Section 2.5.3.25.2
missile_util_eval_poly	Section 2.5.3.26.1
map_get_ammo_entry_from_network_type	Section 2.6.11.2.1

Table 2.5-43: missile_adat_fire Information.

2.5.3.2.3 missile_adat_fly_missiles

This routine flies out all missiles in a flying state. *sight_location* is the location in world coordinates of the gunner's sight. *loc_sight_to_world* is the sight to world transformation matrix used only in this routine. *veh_list* is the vehicle list ID.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
sight_location	VECTOR	/simnet/common/include/protocol/sim_types.h
loc_sight_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h
veh_list	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
Calls		
Function	**Where Described**	
missile_adat_fly	Section 2.5.3.2.4	

Table 2.5-44: missile_adat_fly_missiles Information.

2.5.3.2.4 missile_adat_fly

This routine performs the functions specifically related to the flying of an ADAT missile.

The parameters are represented as follows:

aptr - a pointer to the ADAT missile that is to be flown out.
sight_location - the location in world coordinates of the gunner's sight.
loc_sight_to_world - the sight to world transformation matrix used only in this routine.
tube - the tube from which the missile was launched.
veh_list - the vehicle list ID.

Internal variables are as follows:

mptr - a pointer to the generic aspects of *aptr*.
time - the current time after launch in ticks.
bias - the value of the temporal bias.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
aptr	ADAT_MISSILE	Section 2.5.3.6
sight_location	VECTOR	/simnet/common/include/protocol/sim_types.h
loc_sight_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h
tube	int	Standard
veh_list	int	Standard

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
mptr	MISSILE	Section 2.5.3.2
time	REAL	/simnet/common/include/protocol/sim_types.h
bias	REAL	/simnet/common/include/protocol/sim_types.h

Calls	
Function	**Where Described**
missile_util_eval_poly	Section 2.5.3.26.1
missile_target_los_bias	Section 2.5.3.29.1
missile_target_los	Section 2.5.3.19.1
missile_target_unguided	Section 2.5.3.24.1
missile_util_flyout	Section 2.5.3.27.1
missile_adat_stop	Section 2.5.3.2.5
missile_fuze_prox	Section 2.5.3.1.2
missile_util_comm_check_detonate	Section 2.5.3.25.9

Table 2.5-45: missile_adat_fly Information.

2.5.3.2.4 missile_adat_reset_missiles

This routine puts any flying missile into an unguided state.

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

Table 2.5-46: missile_adat_reset_missiles Information.

2.5.3.2.5 missile_adat_stop

This routine causes all concerned to forget about the missile, releasing the memory for use by other missiles. It should be called when the flyout of any ADAT missile is stopped (whether or not it has exploded). Note that this routine can only be called within this module. *aptr* is a pointer to a missile that is to be stopped.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
aptr	pointer to ADAT_MISSILE	Section 2.5.3.6

Calls	
Function	**Where Described**
missile_fuze_prox_stop	Section 2.5.3.1.3
missile_util_comm_stop_missile	Section 2.5.3.25.7

Table 2.5-47: missile_adat_stop Information.

2.5.3.6 miss_adat.h
(./simnet/release/src/vehicle/libsrc/libmissile/miss_adat.h)

This file contains structure and function declarations that relate specifically to the ADAT missile and are used outside the missile library.

2.5.3.7 miss_hellfr.c

(./simnet/release/src/vehicle/libsrc/libmissile/miss_hellfr.c)

This file contains routines which fly out a missile with the characteristics of a HELLFIRE missile.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"basic.h"
"mun_type.h"
"libmap.h"
"miss_hellfr.h"
"libmiss_dfn.h"
"libmiss_loc.h"

The following define the missile characteristics:

HELLFIRE_ARM_TIME
HELLFIRE_BURNOUT_TIME
HELLFIRE_MAX_FLIGHT_TIME
SPEED_0
THETA_0

The following parameters control flight trajectory brehavior:

SIN_UNGUIDE
COS_UNGUIDE
SIM_CLIMB
COS_CLIMB
SIM_LOCK
COS_LOCK
COS_TERM
COS_LOSE

The following terms set the order of the polynomials used to determine the speed or the cosine of the maximum allowed turn rate of the missile at any point in time.

HELLFIRE_BURN_SPEED_DEG
HELLFIRE_COAST_SPEED_DEG

Coefficients for the speed polynomial before motor burnout:

hellfire_burn_speed_coeff [HELLFIRE_BURN_SPEED_DEG + 1]

Coefficients for the speed polynomial after motor burnout:

hellfire_coast_speed_coeff [HELLFIRE_COAST_SPEED_DEG + 1]

Static function declarations:

missile_hellfire_stop()

2.5.3.7.1 missile_hellfire_init

This routine initializes the state of the missile to indicate that it is available, and sets values that never change. *mptr* is a pointer to the HELLFIRE missile to be initialized.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2

Table 2.5-48: missile_hellfire_init Information.

2.5.3.7.2 missile_hellfire_fire

This routine performs the functions specifically related to the firing of a HELLFIRE missile. The parameters are represented as follows:

mptr - a pointer the the HELLFIRE missile that is to be launched.
launch_point - the location, in world coordinates, from which the missile is launched.
launch_to_world - the transformation matrix of the launch platform to the world.
launch_speed - the speed of the launch platform (assumed to be in the direction of the missile).
tube - the tube from which the missile was launched.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
launch_point	VECTOR	/simnet/common/include/protocol/sim_types.h
launch_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h
launch_speed	REAL	/simnet/common/include/protocol/sim_types.h
tube	int	Standard

Calls	
Function	**Where Described**
vec_copy	Section 2.6.2.59
mat_copy	Section 2.6.2.39.1
missile_util_eval_poly	Section 2.5.3.26.1
missile_util_comm_fire_missile	Section 2.5.3.25.2
map_get_ammo_entry_from_network_type	Section 2.6.11.2.1

Table 2.5-49: missile_hellfire_fire Information.

2.5.3.7.3 missile_hellfire_fly

This routine performs the functions specifically related to the flying of a HELLFIRE missile. *mptr* is a pointer to the HELLFIRE missile to be flown out. *target_location* is the location, in world coordinates, of the target. *time* is the current time after launch in ticks.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
target_location	VECTOR	/simnet/common/include/protocol/sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
time	register REAL	/simnet/common/include/protocol/sim_types.h
Calls		
Function	**Where Described**	
missile_util_eval_poly	Section 2.5.3.26.1	
missile_target_agm	Section 2.5.3.15.1	
missile_util_flyout	Section 2.5.3.27.1	
missile_hellfire_stop	Section 2.5.3.7.4	
missile_util_comm_check_intersection	Section 2.5.3.25.8	
missile_util_comm_check_detonate	Section 2.5.3.25.9	

Table 2.5-50: missile_hellfire_fly Information.

2.5.3.7.4 missile_hellfire_stop

This routine causes all concerned to forget about the missile, releasing the memory for use by other missiles. It should be called when the flyout of any HELLFIRE missile is stopped (whether or not it has exploded.) Note that this routine can only be called within this module. *mptr* is a pointer to the HELLFIRE missile that is to be stopped.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
Calls		
Function	**Where Described**	
missile_util_comm_stop_missile	Section 2.5.3.25.7	

Table 2.5-51: missile_hellfire_stop Information.

2.5.3.8 miss_hellfr.h
(./simnet/release/src/vehicle/libsrc/libmissile/miss_hellfr.h)

This file contains structure and function declarations that relate specifically to the HELLFIRE missile and are used outside the missile library.

2.5.3.9 miss_maverick.c
(./simnet/release/src/vehicle/libsrc/libmissile/miss_maverick.c)

This file contains routines which fly a missile with the characteristics of a MAVERICK missile.

Includes:

"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"basic.h"
"mun_type.h"
"libmap.h"
"libmatrix.h"
"libnear.h"
"miss_maverck.h"
"libmiss_dfn.h"
"libmiss_loc.h"

Defines:
The following define the missile characteristics:

MAVERICK_ARM_TIME
MAVERICK_BURNOUT_TIME
MAVERICK_MAX_FLIGHT_TIME
MAVERICK_LOCK_THRESHOLD
MAVERICK_HOLD_THRESHOLD
SPEED_0
THETA_0

The following parameters control flight trajectory behavior:

SIN_UNGUIDE
COS_UNGUIDE
SIN_CLIMB
COS_CLIMB
SIM_LOCK
COS_LOCK
COS_TERM
COS_LOSE

The following define the possible states of the MAVERICK_MISSILE:

MAVERICK_FREE
MAVERICK_READY
MAVERICK_FLYING

The following terms set the order of the polynomials used to determine the speed or the cosine of the maximum allowed turn rate of the missile at any point in time.
MAVERICK_BURN_SPEED_DEG
MAVERICK_COAST_SPEED_DEG

Coefficients for the speed polynomial before motor burnout:
maverick_burn_speed_coeff [MAVERICK_BURN_SPEED_DEG + 1]

Coefficients for the speed polynomial after motor burnout:
maverick_coast_speed_coeff [MAVERICK_COAST_SPEED_DEG + 1]

Memory for the missiles is declared in vehicle specific code. During initialization, a pointer is assigned to this memory, then all memory issues are handled in this module. *maverick_array* is a pointer to missile memory, and *num_mavericks* is the number of defined missiles.

Static function declarations:
missile_maverick_fly()

2.5.3.9.1 missile_maverick_init

This routine copies the parameters into variables that are static to this module and initializes the state of all the missiles. *missile_array* is a pointer to an array of MAVERICK missiles defined in vehicle specific code. *num_missiles* is the number of missiles defined in *missile_array*..

Parameters		
Parameter	**Type**	**Where Typedef Declared**
missile_array	pointer to MAVERICK_MISSILE	Section 2.5.3.10
num_missiles	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard

Table 2.5-52: missile_maverick_init Information.

2.5.3.9.2 missile_maverick_ready

This routine finds, if possible, a missile that is not being used, puts it in a ready state, clears the target ID, and returns a pointer to it.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
&maverick_array[i]	pointer to MAVERICK_MISSILE	pointer to a missile that is currently available
NULL	pointer to MAVERICK_MISSILE	no free missile was found

Table 2.5-53: missile_maverick_ready Information.

2.5.3.9.3 missile_maverick_pre_launch

This routine is called after a missile has been readied and before it has been launched. It determines if the seeker head can see a target, and if it can see a target, stores its position. The parameters and internal variable are represented as follows:

mvptr - a pointer to the missile that is to be serviced.
launch_point - the location of the missile in world coordinates.
launch_to_world - the transformation matrix of the missile to the world.
veh_list - the vehicle list ID.
target - a pointer to the target vehicle's appearance packet.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mvptr	MAVERICK_MISSILE	Section 2.5.3.10
launch_point	VECTOR	/simnet/common/include/protocol/sim_types.h
launch_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h
veh_list	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
target	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
Calls		
Function	**Where Described**	
near_get_preferred_veh_near_vector	Section 2.5.17.2.4	
missile_target_pursuit	Section 2.5.3.23.1	

Table 2.5-54: missile_maverick_pre_launch Information.

2.5.3.9.4 missile_maverick_fire

This routine performs the functions specifically related to the firing of a MAVERICK missile. The parameters and internal variables are represented as follows:

mvptr - a pointer to the MAVERICK missile that is to be launched.
launch_point - the location, in world coordinates, from which the missile is launched.
launch_to_world - the transformation matrix of the launch platform to the world.
launch_speed - the speed of the launch platform, assumed to be in the direction of the missile.
tube - the tube from which the missile was launched.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mvptr	pointer to MAVERICK_MISSILE	Section 2.5.3.10
launch_point	VECTOR	/simnet/common/include/protocol/sim_types.h
launch_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h
launch_speed	REAL	/simnet/common/include/protocol/sim_types.h
tube	int	Standard

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	successful launch
FALSE	int	unsuccessful launch

Calls	
Function	**Where Described**
vec_copy	Section 2.6.2.59
mat_copy	Section 2.6.2.39.1
missile_util_eval_poly	Section 2.5.3.26.1
missile_util_comm_fire_missile	Section 2.5.3.25.2
map_get_ammo_entry_from_network_type	Section 2.6.11.2.1

Table 2.5-55: missile_maverick_fire Information.

2.5.3.9.5 missile_maverick_fly_missiles

This routine flies all missiles in a ready state. *veh_list* is the vehicle list ID.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_list	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
Calls		
Function	**Where Described**	
missile_maverick_fly	Section 2.5.3.9.6	

Table 2.5-56: missile_maverick_fly_missiles Information.

2.5.3.9.6 missile_maverick_fly

This routine performs the functions specifically related to the flying of a MAVERICK missile. The parameters are as follows:

mvptr - a pointer to the MAVERICK missile that is to be flown out.
veh_list - the vehicle list ID.

The internal variables are as follows:

mptr - a pointer to the generic aspects of *mvptr*.
time - the current time after the launch, in ticks.
target - a pointer to the target appearance packet.
target_location - the location of the target.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mvptr	pointer to MAVERICK_MISSILE	Section 2.5.3.10
veh_list	int	Standard

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
mptr	pointer to register MISSILE	Section 2.5.3.2
time	REAL	/simnet/common/include/protocol/sim_types.h
target	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
target_location	VECTOR	/simnet/common/include/protocol/sim_types.h

Calls	
Function	**Where Described**
missile_util_eval_poly	Section 2.5.3.26.1
missile_target_agm	Section 2.5.3.15.1
near_get_preferred_veh_near_vector	Section 2.5.17.2.4
vec_copy	Section 2.6.2.59
missile_util_flyout	Section 2.5.3.27.1
missile_maverick_stop	Section 2.5.3.9.7
missile_util_comm_check_intersection	Section 2.5.3.25.8
missile_util_comm_check_detonate	Section 2.5.3.25.9

Table 2.5-57: missile_maverick_fly Information.

2.5.3.9.7 missile_maverick_stop

This routine causes all concerned to forget about the missile, releasing the missile memory for use by other missiles. It should be called when the flyout of any MAVERICK missile is stopped (whether or not it has exploded). mvptr is a pointer to the MAVERICK missile to be stopped.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mvptr	pointer to MAVERICK_MISSILE	Section 2.5.3.10
Calls		
Function	**Where Described**	
missile_util_comm_stop_missile	Section 2.5.3.25.7	

Table 2.5-58: missile_maverick_stop Information.

2.5.3.10 miss_maverick.h
(./simnet/release/src/vehicle/libsrc/libmissile/miss_maverick.h)

This file contains structure and function declarations that relate specifically to the MAVERICK missile and are used outside the missile library.

2.5.3.11 miss_stinger.c
(./simnet/release/src/vehicle/libsrc/libmissile/miss_stinger.c)

This file contains routines which fly out a missile with the characteristics of a STINGER missile.

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"basic.h"
"mun_type.h"
"libmap.h"
"libmatrix.h"
"libnear.h"
"miss_stinger.h"
"libmiss_dfn.h"
"libmiss_loc.h"

Defines:
The following define the missile characteristics:
STINGER_BURNOUT_TIME
STINGER_MAX_FLIGHT_TIME
STINGER_LOCK_THRESHOLD

SPEED_0
THETA_0
INVEST_DIST_SQ
FUZE_DIST_SQ

The following constants define the possible states of the STINGER_MISSILE:
STINGER_FREE
STINGER_READY
STINGER_FLYING

Coefficients for the speed polynomial before motor burnout:
stinger_burn_speed_coeff [STINGER_BURN_SPEED_DEG + 1]

Coefficients for the speed polynomial after motor burnout:
stinger_coast_speed_coeff [STINGER_COAST_SPEED_DEG + 1]

Memory for the missiles is declared in vehicle specific code. During initialization, a pointer is assigned to this memory, then all memory issues are handled in this module. *stinger_array* is a pointer to missile memory, and *num_stingers* is the number of defined missiles.

Static function declarations:
missile_stinger_fly()

2.5.3.11.1 missile_stinger_init

This routine copies the parameters into variables static to this module and initializes the state of all missiles. It also initializes the proximity fuse.
The parameters are as follows:
missile_array - a pointer to an array of STINGER missiles defined in vehicle specific code.
num_missiles - the number of missiles defined in *missile_array*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
missile_array	pointer to array of STINGER_MISSILE	Section 2.5.3.12
num_missiles	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
Calls		
Function	**Where Described**	
missile_fuze_prox_init	Section 2.5.3.1.1	

Table 2.5-59: missile_stinger_init Information.

2.5.3.11.2 missile_stinger_ready

This routine finds, if possible, a missile that is not being used, puts it in a ready state, clears the target ID, and returns a pointer to it.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
Return Values		
Return Value	**Type**	**Meaning**
&stinger_array[i]	pointer toSTINGER_MISSILE	a pointer to a missile that is currently available
NULL	pointer to STINGER_MISSILE	no missile is available

Table 2.5-60: missile_stinger_ready Information.

2.5.3.11.3 missile_stinger_pre_launch

This routine is called after a missile has been readied and before it has been launched. It determines if the seeker head can see a target and, if it can see a target, stores its position. Parameters are represented as follows:

sptr - a pointer to the missile that is to be serviced.
lauch_point - the location of the missile in world coordinates.
launch_to_world - the transformation matrix of the missile to the world.
veh_list - the vehicle list ID.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
sptr	pointer to STINGER_MISSILE	Section 2.5.3.12
launch_point	VECTOR	/simnet/common/include/protocol/sim_types.h
launch_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h
veh_list	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
target	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
Calls		
Function	**Where Described**	
near_get_preferred_veh_near_vector	Section 2.5.17.2.4	
missile_target_pursuit	Section 2.5.3.23.1	

Table 2.5-61: missile_stinger_pre_launch Information.

2.5.3.11.4 missile_stinger_fire

This routine performs the functions specifically related to the firing of a STINGER missile. The parameters and internal variables are as follows:

sptr - a pointer to the STINGER missile that is to be launched.
launch_point - the location, in world, coordinates from which the missile is launched.
launch_to_world - the transformation matrix of the launch platform to the world.
launch_speed - the speed of the launch platform (assumed to be in the direction of the missile).
tube - the tube from which the missile was launched.
mptr - a pointer to the particular generic missile pointed to by *sptr*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
sptr	pointer to STINGER_MISSILE	Section 2.5.3.12
launch_point	VECTOR	/simnet/common/include/protocol/sim_types.h
launch_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h
launch_speed	REAL	/simnet/common/include/protocol/sim_types.h
tube	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
mptr	pointer to MISSILE	Section 2.5.3.2
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	successful launch
FALSE	int	unsuccessful launch
Calls		
Function	**Where Described**	
vec_copy	Section 2.6.2.59	
mat_copy	Section 2.6.2.39.1	
missile_util_eval_poly	Section 2.5.3.26.1	
missile_target_intercept_find_poly	Section 2.5.3.17.3	
map_get_ammo_entry_from_network_type	Section 2.6.11.2.1	
missile_util_comm_fire_missile	Section 2.5.3.25.2	

Table 2.5-62: missile_stinger_fire Information.

2.5.3.11.5 missile_stinger_fly_missiles

This routine flies out all missiles in a flying state. *veh_list* is the vehicle list ID.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
veh_list	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	int	Standard
Calls		
Function	**Where Described**	
missile_stinger_fly	Section 2.5.3.11.6	

Table 2.5-63: missile_stinger_fly_missiles Information.

2.5.3.11.6 missile_stinger_fly

This routine performs the functions specifically related to the flying out of a STINGER missile. The parameters are as follows:

sptr - a pointer to the STINGER missile that is to be flown out.
veh_list - the vehicle list ID.

The internal variables are as follows:

mptr - a pointer to the generic aspects of *sptr*.
time - the current time after launch, in ticks.
target - a pointer to the target's appearance packet.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
sptr	pointer to STINGER_MISSILE	Section 2.5.3.12
veh_list	int	Standard

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
mptr	register pointer to MISSILE	Section 2.5.3.2
time	REAL	/simnet/common/include/protocol/sim_types.h
target	pointer to VehicleAppearanceVarient	/simnet/common/include/protocol/p_sim.h

Calls	
Function	**Where Described**
missile_util_eval_poly	Section 2.5.3.26.1
near_get_preferred_veh_near_vector	Section 2.5.17.2.4
missile_target_intercept_pre_burnout	Section 2.5.3.17.1
missile_target_intercept	Section 2.5.3.17.2
missile_target_unguided	Section 2.5.3.24.1
missile_util_flyout	Section 2.5.3.27.1
missile_stinger_stop	Section 2.5.3.11.7
missile_fuze_prox	Section 2.5.3.1.2
missile_util_comm_check_detonate	Section 2.5.3.25.9

Table 2.5-64: missile_stinger_fly Information.

2.5.3.11.7 missile_stinger_stop

This routine causes all concerned to forget about the missile, releasing missile memory for use by other missiles. It also clears the proximity fuse targets. It should be called when the flyout of any STINGER missile is stopped (whether or not it has exploded). *sptr* is a pointer to the STINGER missile that is to be stopped.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
sptr	pointer to STINGER_MISSILE	Section 2.5.3.12
Calls		
Function	**Where Described**	
missile_util_comm_stop_missile	Section 2.5.3.25.7	
missile_fuze_prox_stop	Section 2.5.3.1.3	

Table 2.5-65: missile_stinger_stop Information.

2.5.3.12 miss_stinger.h

(./simnet/release/src/vehicle/libsrc/libmissile/miss_stinger.h)

This file contains structure and function declarations that relate specifically to the STINGER missile and are used outside the missile library.

2.5.3.13 miss_tow.c

(./simnet/release/src/vehicle/libsrc/libmissile/miss_tow.c)

This file contains routines which fly out a missile with the characteristics of a TOW missile.

Includes:

"stdio.h"
"sim_types.h"
"sim_dfns.h"
"basic.h"
"mun_type.h"
"libmap.h"
"libmatrix.h"
"miss_tow.h"
"libmiss_dfn.h"
"libmiss_loc.h"

The following define the missile characteristics:

TOW_BURNOUT_TIME
TOW_MAX_FLIGHT_TIME
TOW_RANGE_LIMIT_TIME

The following terms set the order of the polynomials used to determine the speed or the cosine of the maximum allowed turn rate of the missile at any point in time.

TOW_BURN_SPEED_DEG

TOW_COAST_SPEED_DEG
TOW_BURN_TURN_DEG
TOW_COAST_TURN_DEG

Coefficients for the speed polynomial before motor burnout:
tow_burn_speed_coeff[TOW_BURN_SPEED_DEG+1]

Coefficients for the speed polynomial after motor burnout:
tow_coast_speed_coeff[TOW_COAST_SPEED_DEG+1]

The following are declared as static MAX_COS_COEFF:
tow_burn_turn_coeff
tow_coast_turn_coeff

The following function is declared as static:
missile_tow_stop()

2.5.3.13.1 missile_tow_init

This routine initializes the state of the missile to indicate that it is available, and sets values that never change. *tptr* is a pointer to the TOW missile to be initialized.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tptr	pointer to TOW_MISSILE	Section 2.5.3.14

Table 2.5-66: missile_tow_init Information.

2.5.3.13.2 missile_tow_fire

This routine performs the functions specifically related to the firing of a TOW missile.

The parameters are defined as follows:

tptr - a pointer to the TOW missile to be fired.
launch_point - the location, in world coordinates, from which the missile is launched.
loc_sight_to_world - the sight to world transformation matrix used only in this routine.
launch_speed - the speed of the launch platform (assumed to be in the direction of the missile).
tube - the tube from which the missile was launched.

The internal variable is defined as follows:

mptr - a pointer to the particular generic missile pointed to by *tptr*.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tptr	pointer to TOW_MISSILE	Section 2.5.3.6
launch_point	VECTOR	/simnet/common/include/protocol/sim_types.h
loc_sight_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h
launch_speed	REAL	/simnet/common/include/protocol/sim_types.h
range_to_intercept	REAL	/simnet/common/include/protocol/sim_types.h
tube	int	Standard
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
Calls		
Function	**Where Described**	
vec_copy	Section 2.6.2.59	
mat_copy	Section 2.6.2.39.1	
missile_util_comm_fire_missile	Section 2.5.3.25.2	
missile_util_eval_poly	Section 2.5.3.26.1	
map_get_ammo_entry_from_network_type	Section 2.6.11.2.1	

Table 2.5-67: missile_tow_fire Information.

2.5.3.13.3 missile_tow_fly

This routine performs the functions specifically related to the flying a TOW missile. The parameters are defined as follows:

tptr - a pointer to the TOW missile that is to be flown out.
sight_location - the location, in world coordinates, of the gunner's sight.
loc_sight_to_world - the sight to world transformation matrix used only in this routine.

The internal variables are defined as follows;

mptr - a pointer to the generic aspects of *tptr*.
time - the current time after launch, in ticks.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tptr	pointer to TOW_MISSILE	Section 2.5.3.6
sight_location	VECTOR	/simnet/common/include/protocol/sim_types.h
loc_sight_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
time	REAL	/simnet/common/include/protocol/sim_types.h

Calls	
Function	**Where Described**
missile_util_eval_poly	Section 2.5.3.26.1
missile_util_eval_cos_coeff	Section 2.5.3.26.2
missile_target_level_los	Section 2.5.3.18.1
missile_target_ground	Section 2.5.3.16.1
missile_util_flyout	Section 2.5.3.27.1
missile_tow_stop	Section 2.5.3.13.4
missile_util_comm_check_detonate	Section 2.5.3.25.9
missile_util_comm_check_intersection	Section 2.5.3.25.8

Table 2.5-68: missile_tow_fly Information.

2.5.3.13.4 missile_tow_stop

This routine causes all concerned to forget about the missile, releasing missile memory for use by other missiles. It should be called when the flyout of any TOW missile is stopped (whether or not it has exploded.) Note that this routine can only be called within this module. *tptr* is a pointer to a missile that is to be stopped.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tptr	pointer to TOW MISSILE	Section 2.5.3.14
Calls		
Function	**Where Described**	
missile_util_comm_stop_ missile	Section 2.5.3.25.7	

Table 2.5-69: missile_tow_stop Information.

2.5.3.13.5 missile_tow_cut_wire

This routine sets a flag indicating that the guidance wire to this missile is cut. *tptr* is a pointer to the TOW missile whose wire is to be cut.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
tptr	pointer to TOW_MISSILE	Section 2.5.3.14

Table 2.5-70: missile_tow_cut_wire Information.

2.5.3 14 miss_tow.h
(./simnet/release/src/vehicle/libsrc/libmissile/miss_tow.h)

This file contains structure and function declarations that relate specifically to the TOW missile and are used outside the missile library.

2.5.3.15 targ_agm.c
(./simnet/release/src/vehicle/libsrc/libmissile/targ_agm.c)

This file contains the routines which steer missiles on an air to ground missile trajectory.

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"sim_dfns.h"
"libmatrix.h"
libmiss_dfn.h"

Defines:
SPEED_FACTOR
TERMINAL_DIST_SQ

Declarations:
agm_seek()

2.5.3.15.1 missile_target_agm

This routine sets the target to a vector which will yield an air to ground missile trajectory.

The parameters are defined as follows:

mptr - a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.
target - the location, in world coordinates, which of the missile's target. If *target* is NULL, no target has been designated.
sin_unguide - the sine of the angle that the flight path makes with the horizontal piane when the missile is not guided.
cos_unguide - the cosine of the angle that the flight path makes with the horizontal plane when the missile is not guided.
sin_climb - the sine of the climb angle, the angle that the missile must turn up in a sinle time step in order to climb with a constantly increasing pitch rate.
cos_climb - the cosine of the climb angle.
sin_lock - the sine of the lock angle, the angle between the missile's direction of flight and the direction to the target, which the missile attempts to hold constant.
cos_lock - the cosine of the lock angle.
cos_term - the cosine of the angle which defines the terminal guidance cone, both above and below the target. If the missile is inside either of these cones, the missile will be terminally guided.
cos_lose - the cosine of the angle which sets the field of view of the missile. If the target is outside the missile's field of view, the missile will lose its lock on the target.

Internal variables are defined as follows:

velocity - a pointer to the normalized velocity vector of the missile (the second row of the missile's orientation matrix).
targ_vec - a normalized vector which points from the missile to the target.
traj_vec - a normalized vector which points in the desired flight direction.
sin_targ_sq - the square of the sine of the angle that *targ_vec* makes with the horizontal plane.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
target	VECTOR	/simnet/common/include/protocol/sim_types.h
sin_unguide	REAL	/simnet/common/include/protocol/sim_types.h
cos_unguide	REAL	/simnet/common/include/protocol/sim_types.h
sin_climb	REAL	/simnet/common/include/protocol/sim_types.h
cos_climb	REAL	/simnet/common/include/protocol/sim_types.h
sin_lock	REAL	/simnet/common/include/protocol/sim_types.h
cos_lock	REAL	/simnet/common/include/protocol/sim_types.h
cos_term	REAL	/simnet/common/include/protocol/sim_types.h
cos_lose	REAL	/simnet/common/include/protocol/sim_types.h

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp	register REAL	/simnet/common/include/protocol/sim_types.h
velocity	register pointer to REAL	/simnet/common/include/protocol/sim_types.h
targ_vec	VECTOR	/simnet/common/include/protocol/sim_types.h
traj_vec	VECTOR	/simnet/common/include/protocol/sim_types.h
sin_targ_sq	REAL	/simnet/common/include/protocol/sim_types.h

Calls	
Function	**Where Described**
agm_seek	Section 2.5.3.15.2
vec_sub	Section 2.6.2.65
vec_copy	Section 2.6.2.59
vec_dot_prod	Section 2.6.2.54
vec_scale	Section 2.6.2.64
vec_add	Section 2.6.2.57

Table 2.5-71: missile_target_agm Information.

2.5.3.15.2 missile_agm_seek

This routine sets a new target point so that the missile will fly in a straight line at a specified angle with respect to the horizontal plane.

The parameters are defined as follows:

mptr - a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.

sin_unguide - the sine of the angle that the flight path makes with the horizontal plane when the missile is not guided.

cos_unguide - the cosine of the angle that the flight path makes with the horizontal plane when the missile is not guided.

The internal variable of interest is defined as follows:

traj_vec - the normalized vector which describes the desired trajectory of the missile.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
sin_unguide	REAL	/simnet/common/include/protocol/sim_types.h
cos_unguide	REAL	/simnet/common/include/protocol/sim_types.h

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
temp	register REAL	/simnet/common/include/protocol/sim_types.h
traj_vec	VECTOR	/simnet/common/include/protocol/sim_types.h

Calls	
Function	**Where Described**
vec_scale	Section 2.6.2.64
vec_add	Section 2.6.2.57

Table 2.5-72: missile_agm_seek Information.

2.5.3.16 targ_ground.c
(./simnet/release/src/vehicle/libsrc/libmissile/targ_ground.c)

This file contains the routine that steers the missile to the ground.

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

2.5.3.16.1 missile_target_ground

This routine sets the target to a point at or below the ground directly below the missile. *mptr* is a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
Calls		
Function	**Where Described**	
vec_copy	Section 2.6.2.59	

Table 2.5-73: missile_target_ground Information.

2.5.3.17 targ_intrcpt.c
(./simnet/release/src/vehicle/libsrc/libmissile/targ_intrcpt.c)

This file contains the routines that steer a missile on an intercept trajectory.

Includes:
"stdio.h"
"math.h"
"sim_types.h"
"p_sim.h"
"libmatrix.h"
"libmiss_dfn.h"
"libmiss_loc.h"

Defines:
MAX_DEG_SQ
INTER_TOLERANCE
INTER_MAX_ITERATION

Declarations:
calloc()

2.5.3.17.1 missile_target_intercept_pre_burnout

This routine sets the target to a point which will cause the missile whose motor has not burned out to intercept the target. It does so by manipulating data so that **missile_target_intercept()** will work correctly before motor burnout. The routine **missile_target_intercept()** expects the missile to have a range profile which is described by a single polynomial. The missile model uses different polynomials before and after motor burnout. This routine finds a point in space where the missile would have to be in order to arrive at the point in space where the motor will burn out using the coast phase equations. This point is sent to **missile_target_intercept()**.

The parameters are defined as follows:

mptr - a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.
tptr - a pointer to the vehicle appearance packet of the target vehicle.
burn_range - an array containing the coefficients of a polynomial which yields the missile range before motor burnout, given the time of flight.
burn_time - the time of flight when the motor burns out.
burn_deg - the degree of *burn_range.*
range_1 - an array containing the coefficients of a polynomial which yields missile range after motor burnout, given time of flight.
range_2 - an array containing the coefficients of a polynomial which is the square of that of *range_1.*
deg - the degree of *range_1.*

The internal variables are defined as follows:

delta_range - the difference between the distance that the missile will fly from the current time to the burnout time and the distance it would fly if it were using the coast phase equations.
temp_location - the actual current missile location.
delta_location - the vector from the missile's actual current position to its position if the coast phase equations were used.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2 5.3.2
tptr	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
burn_range	array of REAL	/simnet/common/include/protocol/sim_types.h
burn_time	REAL	/simnet/common/include/protocol/sim_types.h
burn_deg	int	Standard
range_1	array of REAL	/simnet/common/include/protocol/sim_types.h
tange_2	array of REAL	/simnet/common/include/protocol/sim_types.h
deg	int	Standard

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
delta_range	REAL	/simnet/common/include/protocol/sim_types.h
temp_location	VECTOR	/simnet/common/include/protocol/sim_types.h
delta_location	VECTOR	/simnet/common/include/protocol/sim_types.h

Calls	
Function	**Where Described**
vec_copy	Section 2.6.2.59
missile_util_eval_poly	Section 2.5.3.26.1
vec_scale	Section 2.6.2.64
vec_add	Section 2.6.2.57
missile_target_intercept	Section 2.5.3.17.2

Table 2.5-74: missile_target_intercept_pre_burnout Information.

2.5.3.17.2 missile_target_intercept

This routine sets the target to a point which will cause the missile to intercept the target. The parameters are defined as follows:

mptr - a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.
tptr - a pointer to the vehicle appearance packet of the target vehicle.
range_1 - an array containing the coefficients of a polynomial which yields missile range after motor burnout, given time of flight.
range_2 - an array containing the coefficients of a polynomial which is the square of that of *range_1*.
deg - the degree of the polynomial of *range_1*.

The internal variables are defined as follows:

sq_poly - a pointer to an array which will contain the coefficients of the polynomial which yields the square of the distance between the target's current position and the predicted intercept point.
range_0 - the calculated current range.
primary_sq_poly[MAX_DEG_SQ] - the preferred place to store the array pointed to by *sq_poly*.
delta - the vector from the current missile position to the target.
targ_vel - the velocity vector of the target.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
tptr	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
range_1	array of REAL	/simnet/common/include/protocol/sim_types.h
range_2	array of REAL	/simnet/common/include/protocol/sim_types.h
deg	int	Standard

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
temp_var	register REAL	/simnet/common/include/protocol/sim_types.h
sq_poly	pointer to register REAL	/simnet/common/include/protocol/sim_types.h
range_0	REAL	/simnet/common/include/protocol/sim_types.h
primary_sq_poly	array MAX_DEG_SQ of REAL	/simnet/common/include/protocol/sim_types.h
delta	VECTOR	/simnet/common/include/protocol/sim_types.h
targ_vel	VECTOR	/simnet/common/include/protocol/sim_types.h
temp_vec	VECTOR	/simnet/common/include/protocol/sim_types.h
two_deg	int	Standard

Calls	
Function	**Where Described**
missile_util_eval_poly	Section 2.5.3.26.1
f2d_vec_copy	Section 2.6.2.10.1
vec_copy	Section 2.6.2.59
vec_sub	Section 2.6.2.65
vec_dot_prod	Section 2.6.2.54
vec_scale	Section 2.6.2.64
missile_util_eval_newton_raphson	Section 2.5.3.26.3
vec_add	Section 2.6.2.57

Table 2.5-75: missile_target_intercept Information.

2.5.3.17.3 missile_target_intercept_find_poly

This routine finds the range and range squared polynomials for a missile given its base velocity profile and initial speed.

The parameters are defined as follows:

speed_deg - the degree of the velocity profile polynomial.
init_speed - the initial speed of the missile.
speed - an array which contains the coefficients of the velocity profile polynomial.
range - an array which will contain the coefficients of the range profile polynomial.
range_2 - an array which will contain the coefficients of the square of the range profile polynomial.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
speed_deg	int	Standard
init_speed	REAL	/simnet/common/include/protocol/sim_types.h
speed	array of REAL	/simnet/common/include/protocol/sim_types.h
range	array of REAL	/simnet/common/include/protocol/sim_types.h
range_2	array of REAL	/simnet/common/include/protocol/sim_types.h

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

Table 2.5-76: missile_target_intercept_find_poly Information.

2.5.3.18 targ_lev_los.c

(./simnet/release/src/vehicle/libsrc/libmissile/targ_lev_los.c)

This file contains the routine which finds a line of sight target, with provisions for a canted sight.

Includes:

"math.h"
"sim_types.h"
"sim_dfns.h"
"libmatrix.h"
"libmiss_dfn.h"

2.5.3.18.1 missile_target_level_los

This routine finds a point in space to which the missile can steer, which will return it to the line of sight. The missile assumes that the sight is level; therefore, steering commands from a canted sight will not steer correctly. This routine provides for this behavior by generating a leveled coordinate system.

The location of the missile relative to the sight, in sight and level coordinates, is found. The target is a point on what the missile believes to be the line of sight, located at a distance along the line equal to the missile's current position plus the distance it will fly during the next time step. Selecting this point guarantees that, if it is within the turning capabilities of the missile, the missile will not overfly this believed line of sight, and will return almost as quickly as possible. The X and Z values of *rel_sight_location* are essentially errors fed to the missile. It believes these errors to be X and Y values in the level coordinate system. The believed line of sight is, then, the line from the sight location (the origin of both the sight and level coordinate systems) to the point found by subtracting the X and Z values in *rel_sight_location* from the X and Z values in *rel_level_location*. This point is expressed in level coordinates. It is then converted to world coordinates, and saved.

The parameters are defined as follows:

mptr - a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.
sight_location - the location of the sight, in world coordinates.
loc_sight_to_world - a transformation matrix which will transform a vector expressed in sight coordinates to one expressed in world coordinates.

The internal variables are defined as follows:

loc_world_to_sight - the matrix used to convert from world coordinates to sight coordinates. It is the transpose of *loc_sight_to_world*.
level_to_world - the matrix used to convert from a coordinate system whose Y-axis is aligned with the line of sight and X-axis lies in the world XY plane (level coordinates) to world coordinates.
world_to_level - *the matrix* used to convert from world coordinates to level coordinates. It is the transpose of *level_to_world*.
rel_level_location - the location of the missile relative to the sight, in level coordinates.
rel_sight_location - the location of the missile relative to the sight, in sight coordinates.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
sight_location	VECTOR	/simnet/common/include/protocol/sim_types.h
loc_sight_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
loc_world_to_sight	T_MATRIX	/simnet/common/include/protocol/sim_types.h
level_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h
world_to_level	T_MATRIX	/simnet/common/include/protocol/sim_types.h
rel_level_location	VECTOR	/simnet/common/include/protocol/sim_types.h
rel_sight_location	VECTOR	/simnet/common/include/protocol/sim_types.h

Calls	
Function	**Where Described**
mat_level_init	Section 2.6.2.46.1
mat_transpose	Section 2.6.2.51.1
vec_sub	Section 2.6.2.65
vec_mat_mul	Section 2.6.2.56.1
vec_add	Section 2.6.2.57

Table 2.5-77: missile_target_level_los Information.

2.5.3.19 targ_los.c
(./simnet/release/src/vehicle/libsrc/libmissile/targ_los.c)

This file contains the routine which finds a line of sight target.

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

2.5.3.19.1 missile_target_los

This routine finds a point in space to which the missile can steer, which will return it to the line of sight.

The location of the missile relative to the sight, in sight coordinates, is found. The target is a point on the line of sight, located at a distance along the line equal to the missile's current position plus the distance it will fly during the next time step. Selecting this point guarantees that, if it is within the turning capabilities of the missile, the missile will not overfly this believed line of sight, and will return almost as quickly as possible. This point is converted to world coordinates, and saved.

The parameters are defined as follows:
mptr - a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.
sight_location - the location of the sight, in world coordinates.
loc_sight_to_world - a transformation matrix which will transform a vector expressed in sight coordinates to one expressed in world coordinates.

The internal variables are defined as follows:
loc_world_to_sight - the transpose of *loc_sight_to_world*, the matrix used to convert from world coordinates to sight coordinates.
rel_location - the location of the missile relative to the sight, in sight coordinates.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
sight_location	VECTOR	/simnet/common/include/protocol/sim_types.h
loc_sight_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
loc_world_to_sight	T_MATRIX	/simnet/common/include/protocol/sim_types.h
rel_location	VECTOR	/simnet/common/include/protocol/sim_types.h

Calls	
Function	**Where Described**
mat_transpose	Section 2.6.2.51.1
vec_sub	Section 2.6.2.65
vec_mat_mul	Section 2.6.2.56.1
vec_add	Section 2.6.2.57

Table 2.5-78: missile_target_los Information.

2.5.3.20 targ_losbias.c
(./simnet/release/src/vehicle/libsrc/libmissile/targ_losbias.c)

This file contains the routine which finds a line of sight target.

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

2.5.3.20.1 missile_target_los_bias

This routine finds a point in space to which the missile can steer, returning it to the line of sight.

The location of the missile relative to the sight, in sight coordinates, is found. The target is biased from a point on the line of sight. This point is located at a distance equal to the missiles current position plus the distance it will fly during the next time step. Selecting this point guarantees that, if it is within the turning capabilities of the missile, the missile will not overfly the line of sight, and it will return almost as quickly as possible. The amount of bias is input, converted to world coordinates, and saved.

The parameters are defined as follows:

mptr - a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.
sight_location - the location of the sight, in world coordinates.
loc_sight_to_world - a transformation matrix which will transform a vector expressed in sight coordinates to one expressed in world coordinates.
bias_x - the horizontal distance (in meters) that the missile is to be displaced off the line of sight.
bias_z - the vertical distance (in meters) that the missile is to be displaced off the line of sight.

The internal variables are defined as follows:

loc_world_to_sight - the matrix used to convert from world coordinates to sight coordinates. It is the transpose of *loc_sight_to_world*.
rel_location - the location of the missile relative to the sight, in sight coordinates.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
sight_location	VECTOR	/simnet/common/include/protocol/sim_types.h
loc_sight_to_world	T_MATRIX	/simnet/common/include/protocol/sim_types.h
bias_x	REAL	/simnet/common/include/protocol/sim_types.h
bias_z	REAL	/simnet/common/include/protocol/sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
loc_world_to_sight	T_MATRIX	/simnet/common/include/protocol/sim_types.h
rel_location	VECTOR	/simnet/common/include/protocol/sim_types.h
Calls		
Function	**Where Described**	
mat_transpose	Section 2.6.2.51.1	
vec_sub	Section 2.6.2.65	
vec_mat_mul	Section 2.6.2.56.1	
vec_add	Section 2.6.2.57	

Table 2.5-79: missile_target_los_bias Information.

2.5.3.21 targ_point.c

(./simnet/release/src/vehicle/libsrc/libmissile/targ_point.c)

This file contains the routine which causes a missile to fly directly at a vehicle.

Includes:

"sim_types.h"
"libmatrix.h"
"libmiss_dfn.h"

2.5.3.21.1 missile_target_point

This routine steers the missile directly to the target. *mptr* is a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time. *loc* is the x, y, z coordinates of the target.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
loc	VECTOR	/simnet/common/include/protocol/sim_types.h

Calls	
Function	**Where Described**
vec_copy	Section 2.6.2.59

Table 2.5-80: missile_target_point Information.

2.5.3.22 targ_pursuit.c
(./simnet/release/src/vehicle/libsrc/libmissile/targ_pursuit.c)

This file contains the routine which causes a missile to fly directly at a vehicle.

Includes:
"p_sim.h"
"libmatrix.h"
"libmiss_dfn.h"

2.5.3.22.1 missile_target_pursuit

This routine steers the missile directly to the target. *mptr* is a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time. *tptr* is a pointer to a target's vehicle appearance packet.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
tptr	pointer to VehicleAppearanceVariant	/simnet/common/include/protocol/p_sim.h
Calls		
Function	**Where Described**	
vec_copy	Section 2.6.2.59	

Table 2.5-81: missile_target_point Information.

2.5.3.23 targ_unguide.c

(./simnet/release/src/vehicle/libsrc/libmissile/targ_unguide.c)

This file contains the routine which causes a missile to continue to fly in the same direction.

Includes:
"libmatrix.h"
"libmiss_dfn.h"

Defines:
SPEED_FACTOR

2.5.3.23.1 missile_target_unguided

This routine finds a point in space directly in front of the missile. *mptr* is a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2

Calls	
Function	**Where Described**
vec_scale	Section 2.6.2.64
vec_add	Section 2.6.2.57

Table 2.5-82: missile_target_unguided Information.

2.5.3.24 util_comm.c
(./simnet/release/src/vehicle/libsrc/libmissile/util_comm.c)

This file contains routines which provide communication between the missiles, the CIG, and the network. The object is to make this communication transparent to the missiles. A structure array, *missile_comm*, is established to facilitate this communication. Each missile in flight is assigned a missile ID which is an index into this array. The stucture contains an event ID entry which is used to communicate with the rest of the world. Other entries in the structure provide a means of passing information between the missiles and the rest of the world. The number of missiles in flight is limited to the number of elements in *missile_comm*. This is the ONLY place in the missile software where the number of missiles in flight is limited.

Includes:

"math.h"
"sim_types.h"
"sim_dfns.h"
"sim_macros.h"
"mass_stdc.h"
"dgi_stdg.h"
"sim_cig_if.h"
"basic.h"
"libevent.h"
"libimps.h"
"libmatrix.h"
"libmap.h"
"libhull.h"
"libkin.h"
"libmsg.h"
"libnetwork.h"
"libmiss_dfn.h"
"libmiss_loc.h"

Defines:

MISS_EMPTY
MISS_FLYING
MISS_INTERSECTED
MISS_SELF_DETONATED
TAN_10
VIEW_RANGE_2
missile_comm[MAX_MISSILE_ID]

2.5.3.24.1 missile_util_comm_init

This routine sets every entry in missile_comm to MISS_EMPTY, and the event_id to 0.

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
Calls		
Function	**Where Described**	
network_missiles_init	Section 2.1.1.3.1.31.1	

Table 2.5-83: missile_util_comm_init Information.

2.5.3.24.2 missile_util_comm_fire_missile

This routine attempts to find an empty entry in *missile_comm*. If one can be found, it is reserved for the missile in *mptr*, and the missile is given a missile ID. An event ID is obtained as well. The direction and speed of the missile is determined, and the network is informed of a missile launch. If a free entry can not be found, the missile is unable to be launched.

The parameters are defined as follows:

mptr - a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.
ammo_type - an index into the ammunition map which describes the type of missile launched.
distinguished - the distinguished guise.
other - the other guise.
target_id - the intended target of the missile, if any.
target_type - the type of intended target, i.e., vehicle, non-vehicle, or unknown.
fuze - the type of fuse on the missile.
tube - the tube from which the missile was launched.

The internal variables are described as follows:

missile_id - used to find a free entry in *missile_comm*.
d_velocity - the direction and speed of the missile.
velocity - the direction and speed of the missile in floats.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
ammo_type	int	Standard
distinguished	ObjectType	/simnet/common/include/protocol/p_sim.h
other	ObjectType	/simnet/common/include/protocol/p_sim.h
target_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
target_type	int	Standard
fuze	ObjectType	/simnet/common/include/protocol/p_sim.h
tube	int	Standard

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
missile_id	register int	Standard
d_velocity	VECTOR	/simnet/common/include/protocol/sim_types.h
velocity	VelocityVector	/simnet/common/include/protocol/sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	an empty entry in missile_comm can be found
FALSE	int	an empty entry cannot be found.

Calls	
Function	**Where Described**
event_get_eventid	Section 2.6.9.1.2
vec_copy	Section 2.6.2.59
vec_scale	Section 2.6.2.64
d2f_vec_copy	Section 2.6.2.2.1
network_send_missile_fire_pkt	Section 2.1.1.3.1.31.4

Table 2.5-84: missile_util_comm_fire_missile Information.

2.5.3.24.3 missile_util_comm_fly_missile

This routine informs the world of where to fly the missile.

The parameters are defined as follows:

mptr - a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.
chord_start - the position of the missile at the beginning of this time step.
velocity - the velocity vector of the missile, including magnitude and direction.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
chord_start	VECTOR	/simnet/common/include/protocol/sim_types.h
velocity	VECTOR	/simnet/common/include/protocol/sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
missile_id	register int	Standard
Calls		
Function	**Where Described**	
kinematics_range_squared	Section 2.5.8.10.1	
store_traj_chord	Section 2.1.2.2.2.14.3	
map_get_tracer_from_ammo_entry	Section 2.6.11.2.9	
network_send_missile_appearance	Section 2.1.1.3.1.31.2	
missile_util_comm_stop_missile	Section 2.5.3.25.7	

Table 2.5-85: missile_util_comm_fly_missile Information.

2.5.3.24.4 missile_util_comm_intersected_poly

This routine is called by the CIG to record the intersection when the missile hits a polygon. If a missile with the given event ID is found, the intersection is recorded.

The parameters are described as follows:
event_id - the event ID associated with the missile.
soil_type - the type of soil hit.
intersection_point - the location of the intersection.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
event_id	int	Standard
soil_type	int	Standard
intersection_point	pointer to R4P3D	dgi_stdg.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	register int	Standard

Table 2.5-86: missile_util_comm_intersected_poly Information.

2.5.3.24.5 missile_util_comm_intersected_model

This routine is called by the CIG to record the intersection when the missile hits a model. If a missile with the given event ID is found, the intersection is recorded.

The parameters are defined as follows:
event_id - the event ID associated with the missile.
vehicle_id - the vehicle the missile hit.
object_type - the type of object hit (hull or turret).
intersection_point - the location of the intersection.
chord_start - the location of the beginning of the chord.
chord_end - the location of the end of the chord.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
event_id	int	Standard
vehicle_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
object_type	int	Standard
intersection_point	pointer to R4P3D	dgi_stdg.h
chord_start	pointer to R4P3D	dgi_stdg.h
chord_end	pointer to R4P3D	dgi_stdg.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	register int	Standard

Table 2.5-87: missile_util_comm_intersected_model Information.

2.5.3.24.6 missile_util_comm_fuze_detonate

This routine is called by a fuse when a detonation occurs. Information about the detonation that is needed by the network and CIG is determined and recorded.

The parameters are defined as follows:

mptr	- a pointer to the missile.
target_id	- the vehicle id of the target.
miss_pt	- the location of the missile at detonation.
chord_start	- the location, in world coordinates, of the start of the missile chord which caused the detonation, relative to the location of the target at detonation.
chord_end	- the location, in world coordinates, of the end of the missile chord which caused the detonation, relative to the location of the target at detonation.
target_w_to_h	- a matrix which transforms from world coordinates to the target vehicle's hull coordinates.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
target_id	pointer to VehicleID	/simnet/common/include/protocol/basic.h
miss_pt	VECTOR	/simnet/common/include/protocol/sim_types.h
chord_start	VECTOR	/simnet/common/include/protocol/sim_types.h
chord_end	VECTOR	/simnet/common/include/protocol/sim_types.h
targ_w_to_h	T_MATRIX	/simnet/common/include/protocol/sim_types.h

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
id	int	Standard
chord_pt	VECTOR	/simnet/common/include/protocol/sim_types.h

Calls	
Function	**Where Described**
vec_mat_mul	Section 2.6.2.56.1

Table 2.5-88: missile_util_comm_fuze_detonate Information.

2.5.3.24.7 missile_util_comm_stop_missile

This routine instructs the world to stop flying the missile, and frees the space in *missile_comm*. *mptr* is a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
Calls		
Function	**Where Described**	
network_send_non_impact	Section 2.1.1.3.1.33.1	
network_stop_missile_flyout	Section 2.1.1.3.1.31.3	

Table 2.5-89: missile_util_comm_stop_missile Information.

2.5.3.24.8 missile_util_comm_check_intersection

This routine informs a missile of whether it has intersected with a polygon. This routine returns TRUE if the missile has intersected with a polygon and FALSE if it has not. *mptr* is a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	missile detonated
FALSE	int	missile has not detonated

Table 2.5-90: missile_util_comm_check_intersection Information.

2.5.3.24.59 missile_util_comm_check_detonate

This routine informs the rest of the world of a missile detonation. This routine returns TRUE if the missile has detonated and FALSE if it has not. If detonation has occurred, the routine determines whether it was due to a ground impact, a vehicle impact, or a self detonation, and notifies the network and the CIG. If the missile self detonated, the routine also sends the square of the distance from the explosion to the target in the range field. An error message is printed if an invalid detonation was attempted. *mptr* is a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.

The internal variables are defined as follows:

id - the current entry in *missile_comm.*

range_2 - the square of the range from the firing vehicle's current position to the location of the detonation.

flight_path - a vector from the missile's launch position to the location of the detonation.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2

Internal Variables		
Internal Variables	**Type**	**Where Typedef Declared**
id	int	Standard
range_2	REAL	/simnet/common/include/protocol/sim_types.h
flight_path	VECTOR	/simnet/common/include/protocol/sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
TRUE	int	missile detonated
FALSE	int	missile has not detonated

Calls	
Function	**Where Described**
kinematics_range_squared	Section 2.5.8.10.1
vec_sub	Section 2.6.2.65
network_send_vehicle_impact	Section 2.1.1.3.1.70.1
impacts_queue_effect	Section 2.5.15.1.3
network_send_ground_impact	Section 2.1.1.3.1.25.1
vec_dot_prod	Section 2.6.2.54

Table 2.5-91: missile_util_comm_check_detonate Information.

2.5.3.25 util_eval.c

(./simnet/release/src/vehicle/libsrc/libmissile/util_eval.c)

This file contains routines which evaluate and manipulate polynomials to aid in solving problems related to the missiles.

Includes:
"sim_types.h"
"sim_macros.h"
"libmiss_dfn.h"

Defines:
MAX_DER_DEG

Declared:
calloc()

2.5.3.25.1 missile_util_eval_poly

This routine finds and returns the value of a polynomial of any order evaluated at a particular parameter, *param*.

The parameters are defined as follows:

deg - the degree of the polynomial.
coeff - an array containing the coefficients of the polynomial. The coefficients should be stored in the array from the lowest order coefficient to the highest order coefficient.
param - the parameter used to find the value of the polynomial.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
deg	int	Standard
coeff	array of REAL	/simnet/common/include/protocol/sim_types.h
param	REAL	/simnet/common/include/protocol/sim_types.h
Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
result	register REAL	/simnet/common/include/protocol/sim_types.h
Return Values		
Return Value	**Type**	**Meaning**
result	REAL	the value of the polynomial at *param*

Table 2.5-92: missile_util_eval_poly Information.

2.5.3.25.2 missile_util_eval_cos_coeff

This routine finds the value of the three cosines of the maximum turn angle polynomials. They can be of any order.

The parameters are defined as follows:

mptr - a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.

coeff - an array of type MAX_COS_COEFF. This array contains the degrees of the three polynomials used to calculate the cosine of the maximum allowed turn rate in the up, down, and sideways directions, and three arrays with the coefficients of the polynomials.

param - the parameter used to find the value of the polynomials.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2
coeff	pointer to MAX_COS_COEFF	Section 2.5.3.2
param	REAL	/simnet/common/include/protocol/sim_types.h
Calls		
Function	**Where Described**	
missile_util_eval_poly	Section 2.5.3.26.1	

Table 2.5-93: missile_util_eval_cos_coeff Information.

2.5.3.25.3 missile_util_eval_newton_raphson

This routine uses the Newton-Raphson method to find the value of the parameter which will cause the polynomial to have a value of zero. The iteration process continues until the difference between two consecutive solutions is within the prescribed tolerance, or until the maximum number of iterations allowed is exceeded. The result obtained is then returned.

The parameters are defined as follows:
deg - the degree of the polynomial to be solved.
coeff - an array containing the coefficients of the polynomial. The coefficients should be stored in the array from the lowest order coefficient to the highest.
seed - a first guess to the solution.
tolerance - the absolute value of the maximum difference allowed between two consecutive solutions before a result is returned.
max_iter - the maximum number of iterations allowed before returning a result.

The internal variables are defined as follows:
result - the current assumed solution.
delta - the difference between the current and next guess.
der - a pointer to the array containing the coefficients of the derivative of the polynomial.
primary_der[MAX_DER_DEG] - the preferred array in which to store the derivative.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
deg	int	Standard
coeff	array of REAL	/simnet/common/include/protocol/sim_types.h
seed	REAL	/simnet/common/include/protocol/sim_types.h
tolerance	REAL	/simnet/common/include/protocol/sim_types.h
max_iter	int	Standard

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
result	register REAL	/simnet/common/include/protocol/sim_types.h
delta	register REAL	/simnet/common/include/protocol/sim_types.h
der	pointer to register REAL	/simnet/common/include/protocol/sim_types.h
primary_der	array MAX_DER_DEG of REAL	/simnet/common/include/protocol/sim_types.h

Return Values		
Return Value	**Type**	**Meaning**
result	REAL	the value of a parameter that causes the polynomial to be equal to zero
Calls		
Function	**Where Described**	
missile_util_eval_poly	Section 2.5.3.26.1	

Table 2.5-94: missile_util_eval_newton_raphson Information.

2.5.3.26 util_flyout.c

(./simnet/release/src/vehicle/libsrc/libmissile/util_flyout.c)

This file contains the routine which is called by specific missiles to perform the actual flyout functions.

Includes:

"math.h"
"sim_types.h"
"sim_dfns.h"
"libmatrix.h"
"libmiss_dfn.h"
"libmiss_loc.h"

2.5.3.26.1 missile_util_flyout

Given the missile's allowed turn rate and desired position for the current time step and its last normalized velocity vector, the new normalized velocity vector is found and copied into *mptr*. This information, along with the current speed, is used to determine the new missile position. This information is passed on to the CIG and the network. *mptr* is a pointer to a structure of type MISSILE which contains all of the information that describes a missile's state at a given time.

The internal variables are defined as follows:

velocity - a pointer to the missile's last and next normalized velocity vector.
orientation - a pointer to the missile's orientation matrix.
des_traj - the normalized vector from the missile's current position to its desired position, i.e., the desired trajectory.
cos_max_turn - the cosine of the maximum allowed turn angle in the desired turn direction.
cos_des_turn - the cosine of the angle between the last missile velocity and the desired trajectory.
comp_x - the component of *des_traj* in the X-axis of the missile.
comp_z - the component of *des_traj* in the Z-axis of the missile.
old_pos - the location of the missile at the beginning of this time step.

Parameters		
Parameter	**Type**	**Where Typedef Declared**
mptr	pointer to MISSILE	Section 2.5.3.2

Internal Variables		
Internal Variable	**Type**	**Where Typedef Declared**
i	register int	Standard
scale	register REAL	/simnet/common/include/protocol/sim_types.h
velocity	pointer to register REAL	/simnet/common/include/protocol/sim_types.h
orientation	T_MAT_PTR	/simnet/common/include/protocol/sim_types.h
des_traj	VECTOR	/simnet/common/include/protocol/sim_types.h
cos_max_turn	REAL	/simnet/common/include/protocol/sim_types.h
cos_des_turn	REAL	/simnet/common/include/protocol/sim_types.h
comp_x	REAL	/simnet/common/include/protocol/sim_types.h
comp_z	REAL	/simnet/common/include/protocol/sim_types.h
old_pos	VECTOR	/simnet/common/include/protocol/sim_types.h

Return Values		
Return Values	**Type**	**Meaning**
TRUE	int	The time step has been incremented
FALSE	int	This time step was the last one allowed

Calls	
Function	**Where Described**
vec_sub	Section 2.6.2.65
vec_dot_prod	Section 2.6.2.54
vec_copy	Section 2.6.2.59
vec_scale	Section 2.6.2.64
vec_add	Section 2.6.2.57
missile_util_comm_fly_missile	Section 2.5.3.25.3

Table 2.5-95: missile_util_flyout Information.

2.5.3.27 util_init.c

(./simnet/release/src/vehicle/libsrc/libmissile/util_init.c)

This file contains the routine called to perform the initializations required by the missiles.

Includes:
"libmiss_loc.h"

2.5.3.27.1 missile_util_init

This routine calls the initialization routine in the communications module.

Calls	
Function	**Where Described**
missile_util_comm_init	Section 2.5.3.25.1

Table 2.5-96: missile_util_init Information.