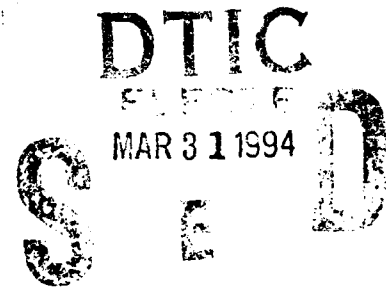AD-A277 803

IDA DOCUMENT D-1452

THE SMART MINEFIELD SIMULATOR USER'S GUIDE
AND ALGORITHM DESCRIPTION

R. E. Schwartz, *Project Leader*

DTIC

MAR 3 1 1994

94 3 31 086

December 1993

Prepared for
Advanced Research Projects Agency

DTIC

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311-1772

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE December 1993	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE

The Smart Mine Simulator User's Guide and Algorithm Description

5. FUNDING NUMBERS

MDA 903 89 C 0003

TASK A-117

6. AUTHOR(S)

R.E. Schwartz, R.W. Carpenter, M.M. Stahl

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

Institute for Defense Analyses
1801 N. Beauregard St.
Alexandria, VA 22311-1772

8. PERFORMING ORGANIZATION REPORT NUMBER

D-1452

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

Advanced Research Projects Agency
Virginia Square Plaza
3701 N. Fairfax Drive
ARPA/LSO
Arlington, VA 22203

Director, FFRDC Programs
1801 N. Beauregard St.
Alexandria, VA 22311-1772

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release. Distribution Unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Smart Mine Simulator (SMS) is a computer simulation that runs on two UNIX workstations and operates in the SIMNET/BDS-D distributed simulation environment. It simulates smart antiarmor mines, two variations of smart antihelicopter mines, and conventional antiarmor mines, enabling these mines to participate in SIMNET exercises for analytic, training, demonstration, or other purposes. This document describes the SMS structure, its algorithms for simulating mines, and how to install and use it. The document is intended to support both the planning of distributed simulation exercises and the installation and operation of the SMS on simulation networks.

14. SUBJECT TERMS

SIMNET, BDS-D, DIS, smart mines, wide area mines, WAM, antihelicopter mines, AHM, simulation, simulator

15. NUMBER OF PAGES

72

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAME AS REPORT

UNCLASSIFIED

IDA DOCUMENT D-1452

THE SMART MINEFIELD SIMULATOR USER'S GUIDE AND ALGORITHM DESCRIPTION

R. E. Schwartz, *Project Leader*

R. W. Carpenter
M. M. Stahl

Accesion For		
NTIS CRA&I		☑
DTIC TAB		☑
Unannounced		☐
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

December 1993

IDA

INSTITUTE FOR DEFENSE ANALYSES

UNCLASSIFIED

PREFACE

This document describes the Smart Mine Simulator (SMS), a simulator of smart and conventional mines that operates in the SIMNET distributed simulation environment. The SMS was developed under a joint task[1] from the Advanced Research Projects Agency, and the Armament Research and Development Center (ARDEC) of the Army.

The authors wish to thank the IDA technical reviewers, Dr. David L. Randall, Director, System Evaluation Division, Dr. Frederic A. Miercort, and Mr. Frederick E. Saxe, for their helpful suggestions.

CONTENTS

APPENDIXES

LIST OF FIGURES

ix

LIST OF TABLES

GLOSSARY

AAM	Antiarmor Mine
ACU	Autonomous Control Unit
ADST	Advanced Distributed Simulation Technology
AHM	Antihelicopter Mine
APC	Armored Personnel Carrier
ARDEC	(U.S. Army) Armament Research, Development, and Engineering Center
ARPA	Advance Research Projects Agency
BDS-D	Battlefield Distributed Simulation - Developmental
CIG	Computer Image Generator
CPA	Closest Point of Approach
DIS	Distributed Interactive Simulation
DSB	Defense Science Board
FASCAM	Family of Scatterable Mines
IFV	Infantry Fighting Vehicle
IR	Infrared
MCC	Management, Command, and Control
OOP	Object-Oriented Programming
PDU	Protocol Data Unit
RF	Radio Frequency
SAF, SAFOR	Semi-automated Forces
SAO, SAObject	Stand-alone Object
SGI	Silicon Graphics, Incorporated
SIMNET	Simulator Networking
SMS	Smart Minefield Simulator
SSM	Smart Standoff Mine
STRICOM	(U.S. Army) Simulation, Training, and Instrumentation Command
VAP	Vehicle Appearance PDU
WAM	Wide Area Mine

I. INTRODUCTION

The Smart Mine Simulator (SMS) allows a human operator to employ mines during a simulated battle occurring on the Battlefield Distributed Simulation - Developmental (BDS-D) network. It allows the user to place minefields on the battlefield using a plan-view (two dimensional) display of the terrain, and simulates the actions of mines in the presence of vehicles generated by other simulators. It models four different mine types: conventional buried landmines, smart antiarmor mines (AAMs), and two variations of smart antihelicopter mines (AHMs).

The purpose of the SMS is to represent the effects of mines in land warfare, and particularly the potential effects of developments such as smart standoff mines (SSMs) and mine command and control (C^2) that may greatly extend the effects and uses of mines. The simulator software uses object-oriented design and implementation, which will allow the simulation to be extended easily to include variations of the currently implemented mine simulations, new mine simulations, new command and control options, or new capabilities such as simulating countermeasures.

The SMS does not itself simulate a complete battle: it does not simulate vehicles or weapon systems other than the listed mine types. Other simulators, such as manned vehicle simulators or Semiautomated Forces (SAF), are needed to conduct a meaningful simulation exercise. The following sections of this Introduction briefly describe mines, SIMNET and BDS-D, and the purpose and scope of the SMS development project, but does not constitute a reference for BDS-D or other simulators. This material is intended to make the following SMS description understandable to all readers, but is not sufficient to enable the reader to plan, set up, or conduct a BDS-D exercise.

Chapter II provides an overview of the SMS, including discussions of its design, capabilities, and missing capabilities.

Chapter III is a user's guide. It presumes some knowledge of both SIMNET and of UNIX.

1

Chapter IV describes the way the mines are modeled, and is intended to allow the analyst to understand the capabilities and limitations of the SMS as well as to understand its results.

A. MINES

Antitank landmines, consisting of buried high explosives with pressure fuzes, were developed shortly after the advent of armored vehicles. Subsequent developments of these buried conventional mines have most notably consisted of new fusing mechanisms (such as tilt-rod fuzes) that trigger an explosion when any part of a tank, not just the treads, passes over a mine. In the 1970s scatterable mines were developed. These lightweight, magnetically fuzed surface mines can be delivered by aircraft, artillery, and ground vehicles.

Pressure fuzed mines function under the track (or wheel) of the activating vehicle. Damage consists, for tanks, of broken track and suspension system components, depending on the weight of the explosive charge. Damaged tanks can be repaired and returned to action by organizational or direct support maintainence. The crew is generally uninjured, but may be at risk from covering fire as the tank is immobilized.

Influence fuzed mines will fire a slug through the bottom of the tank (about 60% of the time) or break the track (about 40% of the time). When the slug perforates the bottom it causes catastrophic damage, and the crew will not usually survive.

Conventional and scatterable mines are usually deployed in minefields, a concentration of mines in a given area. (Random emplacement in a large area has also been done.) Because tanks must pass directly over these mines to trigger them, large numbers are required for an extensive minefield. Conventional mines are heavy and must be buried, so that a considerable logistical and manpower burden is incurred in emplacing a minefield. Scatterable mines are much lighter and more easily emplaced but are still needed in large numbers. For example, U.S. doctrine calls for emplacing one mine per meter of minefield front.

Historically, mines have been effective weapons. According to a 1986 Defense Science Board (DSB) Summer Study, the percentages of tank losses credited to mines in World War II, the Korean war, and Vietnam were 18 to 34 percent, 56 percent, and 69

percent, respectively.[2] The DSB study concluded that the United States is not placing sufficient emphasis on mine or countermine warfare. This was attributed to lack of training and organizational support and also to the lack of tools to quantify mine effectiveness adequately. One of the reasons that landmine effectiveness is difficult to quantify is that the primary tactical uses of landmines are to deny territory and delay movement, effects that are more difficult to evaluate than are the losses of forces.

A new type of antitank mine is now under development, called a Wide Area Mine or WAM. The distinguishing WAM concept is that it is effective at a distance, with the version under development having a range of about 100 meters. It detects and tracks a tank (or other vehicle) with acoustic and seismic sensors, and fires a projectile called a sublet over the path of the tank. The sublet has an infrared sensor that detects the tank's engine and an explosively formed penetrator that it fires at the tank.

Another type of mine that has been proposed is the antihelicopter mine, or AHM. Like the WAM, the AHM detects and tracks its target using acoustic sensors, and attacks at a distance. There are two versions now under development, the direct fire AHM that fires directly at the helicopter and the sublet-launching (or indirect fire) AHM that like the WAM fires a sublet that itself detects and fires at the target.

Both WAM and AHM are being designed with a remote control system using a secure RF link, enabling active command and control of the mines. C^2 concepts are important issues for both types of mines. The new mine types and C^2 potential will enable new deployment options and tactical missions. For example WAM might be used to mine roads and other areas where conventional mines are difficult to use, and the ability to turn mines off may allow their use in areas that friendly forces will traverse.

B. SIMNET AND BDS-D

SIMNET was an Advanced Research Projects Agency (ARPA) program that developed networks of manned simulators, primarily as tactical training systems for armor (tank and armored personnel carrier) crews. The SIMNET program has been completed, and the networks and simulators that were developed are operational at sites in the continental United States and Europe. The simulators and associated equipment have been transferred to the Army, which operates them as training systems and as the experimental BDS-D network under STRICOM's Advanced Distributed Simulation Technology (ADST)

---

[2] *Defense Science Board Summer Study on Mine/Countermine Warfare* (U), February 1987, SECRET.

3

program. This technology is now commonly referred to as Distributed Interactive Simulation (DIS) and is being further developed in several ARPA and Army programs.

The strength of DIS is that a battle outcome is determined by the decisions and actions of human participants, in addition to technical parameters such as weapon effects. These decisions are based on data presented to operators in ways simulating actual presentation, such as views of the battlefield or commands from superiors. The intent is to simulate *tactical interactions* among vehicles and crews in ways that cannot be achieved by other (constructive) computer simulations.

In addition to armored vehicle simulators, a number of helicopter and a few other types of simulators have been developed. Each crew member for these vehicles is present in a simulator and must perform most of the functions that he would perform in a real vehicle. The simulators are simpler and less expensive than those designed to train crews in the operation of vehicles. Thus a tank simulator has displays for the vision blocks of a tank and realistic controls for maneuver and weapons fire, but only static pictures for many gauges and dials. The visual displays use Computer Image Generators (CIGs) to show digitized terrain, cultural features, and simulated effects (vehicles, explosions, etc.).

The vehicle simulators are connected with a computer network, with messages called Protocol Data Units (PDUs) sent between vehicles to simulate their interactions. The protocol is designed to minimize the communication overhead, with each simulator maintaining a database of its picture of the battlefield. For example each simulator updates its estimate of the position of other vehicles using dead reckoning (based on the latest known true position and velocity), so that new data need be sent only when the true position deviates from the dead reckoning estimate by more than the allowable error margin. The protocol is also designed to retain realism while allowing simulators to have imperfect knowledge of each other's state. For example a vehicle that fires at another determines whether it scores a hit, as the detection and aiming computations are based on the firing vehicle's knowledge of the target. But the damage that results from an impact is determined by the target, as it knows its internal state.

The network includes nodes to support simulations, in addition to the combat vehicle simulators. A Management, Command and Control (MCC) system simulates a battalion Tactical Operations Center, artillery and close air support, and resupply depots and vehicles. A Stealth or Magic Carpet vehicle can observe all aspects of a simulation without interacting with other elements, and a DataLogger or TableLogger collects data.

4

Semi-Automated Forces (SAF) also have been developed. Using this system, one commander can control many simulated vehicles without using individual simulators and crews. SAF vehicles cannot be distinguished from manned simulator vehicles in their visual representation, and the intent has been to program behavior that cannot readily be distinguished from manned simulator behavior. SAF was developed primarily to represent enemy forces without using crews trained in enemy doctrine, to extend the size of a simulation beyond the available simulator and crew limits, and to train company or higher level commanders without the need for many fully crewed simulators. SAF is now limited in the types of systems it represents and the realism of its forces' behavior, but, as with manned simulators, current programs are extending and enhancing it.

In addition to its use as a training system, BDS-D has been used to a limited extent as a tool for weapons systems analysis and development and conceptually could be used for a variety of analytic purposes.

C. PROJECT HISTORY AND PURPOSE

IDA has carried out several tasks studying landmines, AHMs, and mine C^2, and has also participated in SIMNET development in recent years. In FY 1989 an IDA Central Research Project[3] identified landmines as candidate weapon systems for simulation and evaluation in SIMNET, primarily because:

- There was DoD interest and IDA tasking relating to landmine analyses.

- Many of the issues regarding mine effectiveness are difficult to analyze using traditional methods, primarily because they involve human decision-making and tactical maneuver on a battlefield (which are incorporated in SIMNET).

- The simulation of landmines in SIMNET would be relatively easy, as it would not require a manned simulator.

Following this report, IDA continued to develop the landmine simulator concept and in the spring and summer of 1991 worked with a Loral Corporation employee (Paul Monday) at Fort Knox to develop a very simple, proof-of-principle simulator. Late in 1991 ARPA began this two-year program, partly funded by ARDEC, to develop the SMS and use it for minefield analyses. The initial tasking was to IDA to specify the simulator design, and to Loral (through the STRICOM ADST program) to implement the software.

---

[3] *Applicability of SIMNET to Evaluation of Military Systems*, IDA Document D-640, August 1989, UNCLASSIFIED.

A preliminary version of the SMS was completed in the fall of 1992, and has been used for preliminary analyses and to develop specifications for later versions.

The first releasable version of the SMS (version 1.0) has been developed at IDA during the second year of the program, and is documented in this report. It has not yet been used for mine analyses, but has been tested at IDA, demonstrated at IDA and also at the AUSA convention in October, 1993, and is being installed at ARDEC and other sites.

The SMS development is being continued under new tasking. Section II-H below descitbes current limitations and potential improvements.

II. OVERVIEW

The SMS consists of two executable programs, a front-end or user interface program called the **smsgui** and the back-end or minefield simulation program called the **smshost**. The smsgui uses a graphical interface (similar to that used by the SAF, and based on it) to emplace minefields and perform limited minefield command and control. The smshost implements the simulation of mines and their interactions with other SIMNET entities participating in an exercise.

Both the smshost and the smsgui are SIMNET entities and use SIMNET 6.6.1 protocols to communicate with each other and (in the case of the smshost) with other simulators. In addition to using previously defined PDUs in both the Simulation and Data Collection protocol families, a new family of SMS-specific PDUs has been defined for the emplacement of minefields and the sending of mine appearance and data collection PDUs. These PDUs are now experimental, but have been defined consistently with the existing protocol families so that they could be incorporated into the existing SIMNET protocols at some future time.

This chapter describes the smshost. This document does not contain a comparable description of the smsgui (although the following chapter describes its capabilities, and how to use it as well as how to use the smshost).

A. GENERAL

The SMS models individual mines and their battlefield interactions with vehicles. The interactions vary with the mine type, but in general include detecting, tracking, and attacking targets. Consistent with BDS-D concepts, a hit target (not the SMS) assesses the damage it sustains. The SMS simulates a conventional mine, a smart antitank mine (modeled after the WAM now under development), and two types of AHM (direct fire and indirect fire). It also allows minefields to be turned on and off using commands from the user interface.

The SMS has been designed to allow the addition of other objects within a minefield including other mine types, stand-alone sensors, control units, and radio relays, as well as other landmine command and control options. An object-oriented approach

7

facilitates the use of unique logic modules for different mine types and functions, rather than (for example) using generalized functions with parameters to distinguish among logic options for different mine types.

The SMS is in some ways more like SAF than existing BDS-D manned simulators, as it simulates multiple independent entities and does not require individual human control of each entity. Each minefield component operates independently in terms of its interactions with targets. Conceptually there is not an overall flow of logic of the form "the SMS does this, then this, etc." Rather, there are independent entities that have their own defining logic, and a simulation manager that controls how and when each module is invoked.

B. ARCHITECTURE

The SMS uses an object-oriented design and is written in an object-oriented language, C++. For those unfamiliar with object-oriented programming (OOP), the programming language provides support for the concept that individual objects (e.g., mines) have certain data associated with them (e.g., location, state, list of vehicles currently being tracked) and further that each object behaves in a way defined for its class of objects (e.g., one WAM behaves like other WAMs, but AHMs can behave differently).

OOP objects can correspond directly to real-world entities, or can be software constructs that have more to do with the implementation approach than with the systems being represented. Vehicles are objects within the SMS that correspond to vehicles simulated externally, by other simulators. The objects simulated by the SMS that correspond to real-world entities include minefields, mines, sensors, control units, munitions, and sublets. Mines and sublets are the essential objects modeled in the SMS, and are called Stand-Alone Objects (SAObjects). Each SAObject contains a sensor, control unit, and munition that work together to perform most of the actions of the SAObject. A minefield is a collection of mines that in the real world is an abstract concept (saying that two mines are in the same minefield does not affect the behavior of either) and in the SMS exists partly for the logical correspondence with real-world ideas and partly as a programming convenience (it keeps track of mines, and allows all the mines in the minefield to be turned on or off with a single command).

Other SMS objects control various aspects of the simulation. The most important are the simulation manager, the network interface, the vehicle manager, and the minefield manager. Figure 1 shows a hierarchy for these classes of objects that at the lower levels

8

can best be thought of as a containment hierarchy (a mine is contained in a minefield) but at the top level is perhaps best thought of as a control hierarchy (the simulation manager controls the network interface). A perhaps unexpected feature of this diagram is that sublets are controlled directly by the simulation manager, and are not contained in minefields. As explained below, this is to ensure that sublets receive priority processing.
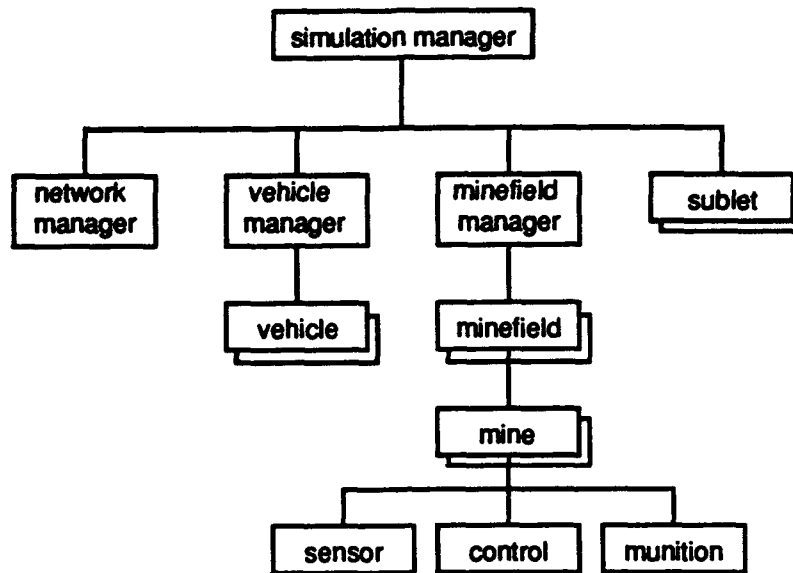
Figure 1. Object Control/Containment Hierarchy

C. SIMULATION MANAGER

The primary functions of the simulation manager are to:

- Initialize the simulation and read inputs.

    The simulation inputs from the invocation command line and input files are described below in the User's Guide, Chapter III.

- Control time.

    BDS-D is a real-time simulation, so the SMS uses the computer's internal clock to control the timing of simulated events. It operates in cycles of 1/10 second duration. At the beginning of each cycle, the simulation manager increments the time defining the end of the current cycle. During the cycle, other objects perform actions needed to update themselves to this time.

- Control the subordinate objects.

    The simulation manager invokes in order the network interface, the vehicle manager, each flying sublet, and the minefield manager. In order to give

9

priority to the most needed tasks and provide graceful degradation if the processor is overloaded, minefield processing is interrupted if it is not completed by the end of a cycle. In the next cycle it resumes where it was interrupted.

D. NETWORK MANAGER

The network manager handles all incoming and outgoing PDUs that interface the SMS host with SIMNET. Outgoing PDUs created within the SMS are sent as soon as they are created. For example, when a mine needs to send a message that a vehicle has been hit, it calls the appropriate network manager routines to create and send the PDU.

Incoming PDUs are processed at the beginning of each cycle. Each PDU that has arrived since the last cycle is processed by reading the header information to determine the nature of the PDU, and forwarding it to the SMS object that handles that PDU. For example, Vehicle Appearance PDUs are forwarded to the vehicle manager, and minefield emplacement PDUs are forwarded to the minefield manager. Tables 1 and 2 show the PDUs that are handled by the SMS host. The contents of SIMNET PDUs are described in the BBN document *The SIMNET Network and Protocols*,[4] and the contents of the new PDUs defined for the SMS are described in Appendix B below.

Table 1. Incoming PDUs

	PDU Kind	From	Handling Object
	vehicle appearance	other simulator	vehicle manager
	status query	other simulator	simulation manager
	deactivate request	other simulator	vehicle manager
+	emplacement	user interface (smsgui)	minefield manager
+	radio command (on/off)	user interface (smsgui)	minefield

+ New type of PDU for the SMS.

---

[4] *The SIMNET Network and Protocols*, BBN Systems and Technologies Report 7627, June 1991.

Table 2. Outgoing PDUs

PDU Kind	Creator	Destination	When Sent
simulation status	simulation manager	broadcast	every 5 minutes
status response	simulation manager	query sender	in response to query
exercise status	simulation manager	query sender	in response to query
minefield status	minefield	broadcast	every 5 minutes
+ SAObject status	SAObject	broadcast	every 5 minutes
minefield appearance	minefield	broadcast	every 30 seconds
+ SAObject appearance	SAObject	broadcast	every 30 seconds
vehicle appearance	sublet	broadcast	every 1/10 sec during flyout
weapon fire	mine, sublet	broadcast	at mine or sublet firing
impact	mine, sublet	target or broadcast	at mine or sublet impact

+ New type of PDU for the SMS.

E. VEHICLES

Simulators that simulate vehicles periodically send Vehicle Appearance PDUs (VAPs) describing the vehicle's type, location, velocity, and other characteristics. When a VAP is received the network manager sends it to the vehicle manager. The vehicle manager maintains a vehicle list, and when an incoming VAP corresponds to a vehicle on the list the VAP is used to update its data. When a VAP is received for a vehicle not already listed, a vehicle object is created and added to the list. After all incoming PDUs are processed at the beginning of each cycle, each vehicle object is called to dead reckon to where it will be at the end of the current cycle.

A vehicle is deleted and removed from the list when a deactivate request PDU is received for the vehicle, or when the timeout period has elapsed without receiving a VAP for the vehicle.

F. MINEFIELDS

The minefield manager has a list of minefields. When the SMS operator creates a minefield, an emplacement PDU is sent from the smsgui to the smshost. Within the smshost it is passed to the minefield manager, which creates the minefield and adds it to the list.

During each cycle the minefield manager is called to update the minefields, and as many minefields as possible are updated each cycle. Minefields that cannot be updated during a cycle are the ones processed first during the next cycle. The purpose for this is to provide a smooth degradation as the SMS host becomes overloaded. Most mines need not

11

be updated every 1/10 second, but incoming PDUs should all be processed and flying sublets need to keep up to date.

Each minefield has a list of its mines, and performs functions such as processing commands (on/off) by routing them to each mine. Updating the minefield consists of creating a list of vehicles within sensor range of the minefield and passing this list to each mine to perform its update.

Minefields periodically send Minefield PDUs, which are the counterpart of VAPs, and status PDUs which provide additional data about the minefield.

G. STAND-ALONE OBJECTS

SAObjects comprise a family of classes, as shown in Figure 2, that includes all mines and the sublets that they fire. The mine classes include direct fire AHMs (cAhmDir), indirect fire (or sublet launching) AHMs (cAhmInd), conventional mines (cConv), and smart antitank mines modeled after WAMs (cWAM). The sublet classes include those fired by the indirect fire AHMs (cSubletAhmInd) and those fired by WAMs (cSubletWAM). Each SAObject contains a sensor, a control unit, and a munition component. Each of these component types is specialized for a particular SAObject type, so that there are six classes each of sensors, control units, and munitions corresponding to the six actual SAObject classes.[5]
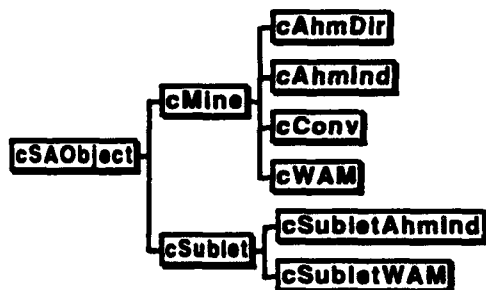
Figure 2. SAObject Class Tree

---

[5] The six actual SAObject classes are those on the right side of Figure 2. The classes cSAObject, cMine, and cSublet are abstract classes, and no objects of these classes exist. Similarly, there are cSensor, cControl, and cMunition abstract classes that are base classes for those that are actually created in the simulation. For those unfamiliar with OOP, this is a way to treat objects similarly and to provide them with some common characteristics, while allowing specialized characteristics when needed.

During each cycle, each SAObject is called to update itself (unless the simulation is overloaded, as described above). This update consists of checking nearby vehicles (a list is provided by the minefield) and taking action as necessary. Chapter IV explains how the SAObjects and components are modeled.

SAObjects periodically send SAObject appearance PDUs, which are the counterpart of VAPs for vehicles, and status PDUs, which provide additional data about the SAObject.

In the SIMNET protocols there are standard sequences or conceptual models for sending fire and impact PDUs when a vehicle is attacked, with the normal sequence being a fire PDU followed by an impact PDU. In most cases the actions of mines in the SMS do not correspond directly to these conceptual models, so that the sequence of sending the fire and impact PDUs has been adapted. Conventional mines send an impact PDU but no fire PDU, and mines that have sublets send a fire PDU when the sublet is launched, a fire PDU when the sublet attacks, and an impact PDU.

H. LIMITATIONS AND POTENTIAL IMPROVEMENTS

Not all of the capabilities that were originally envisioned for the SMS have been implemented in the current version, and in any case it is envisioned as a research tool that can be extended as new requirements (e.g., new C^2 ideas) are defined. Some of the improvements that may be made in future versions (that in some cases reflect limitations in the current version) include:

- DIS protocol

  The current version uses the SIMNET version 6.6.1 protocol. A goal for the SMS is to use it in environments (such as BattleLabs) that will require the DIS protocol.

- Standarization of SMS PDUs

  The new PDUs defined for the SMS are not now recognized by other simulators, which precludes, for example, visually displaying mines. The goal is that after a period of review and development new mine PDUs will be added to the DIS standard.

- Munition types

  The fire and impact PDUs sent by the SMS specify burst descriptors that have been defined for other weapons (e.g., an artillery shell burst). The reason for this is that other simulators have effects tables to assess the damage they incur,

13

and specifying a type not in the current tables would result in no damage. Defining new munition types for mines will require developing effects tables and adding them to the software for other simulators.

- Mine types

  Other mine types, or variations on the existing mine types, may be needed. For example, only a single generic conventional mine is now modeled.

- Countermeasures

  The current version has no countermeasure capabilities.

- C^2 options

  The only C^2 capability available now is on/off. Possible extensions include the development of a Minefield Control Unit simulator to provide better direct operator control, an Autonomous Control Unit (ACU) to provide autonomous control within a minefield, or cooperative control among the mines in a minefield.

- Intelligent Minefield concepts

  In addition to new C^2 concepts, the Intelligent Minefield program is analyzing concepts for smart radio relays (gateways) to communicate with minefields and special vehicles to help lay, pick up, or control minefields.

- User interface (smsgui) improvements

  Improvements might include additional emplacement options, allowing countermeasures, and correcting some of the difficulties described in the User's Guide.

- Test capability

  The current SMS version has a test mode that allows it to run independently of other simulations, using internally generated vehicles. This mode could be extended. Possible benefits might include using it to help plan manned exercises, or using it for preliminary analyses (reducing the number of variations needed for manned exercises).

- Force options

  The current version requires that minefields are part of the distinguished force (i.e., U.S. force).

- Emplacement by other simulators

  Emplacement PDUs could conceptually be sent by other simulators (e.g., vehicles with mine dispensers), but this will require defining a mechanism to establish a link with the smshost and changes to the other simulators.

14

- Miscellaneous software enhancements

  A number of minor improvements have been identified.

III. USER'S GUIDE

A. INTRODUCTION

The SMS consists of two executable programs: **smsgui** and **smshost**. The smsgui is a graphical user interface used to emplace mines and perform minefield command and control. The smshost simulates mines.

Both programs are SIMNET applications and use SIMNET protocols to communicate with each other and with other simulators. In addition to the PDUs defined in the SIMNET 6.6.1 protocol, a new SMS set of five PDUs has been defined. These PDUs are designed to be consistent with the SIMNET protocols, and are sent and received using the standard SIMNET association, transaction, and datagram services. The SMS PDUs are described in Section D below and in Appendix B.

In normal operation the two programs are run on separate computers, along with other simulators participating in an exercise. Each can be run in a stand-alone mode, without interacting with each other or with other simulators. They cannot be run on the same machine during an exercise.

The remainder of this section provides an overview of the two programs and describes how to execute them. Typographic conventions are shown in Table 3.

Table 3. Typographic Conventions

Type Style	Used For
bold	Commands the user will enter. These must be typed exactly as shown and followed by pressing the RETURN key.
italics	Parameters the user will enter. The range of options for each parameter will be described in the text.
[...]	Optional commands or parameter(s). Ellipsis indicates more than one item may be entered.
Large Bold	Menu titles and menu items.

B. THE USER INTERFACE PROGRAM

1. Overview

The SMS user interface program, smsgui, provides a two-dimensional display of terrain and mines. The minefield operator or analyst will use the interface to emplace a variety of mine types in one or more fields and perform minefield command and control.

Four kinds of mines are supported: a generic conventional mine triggered by vehicle proximity, a sublet-launching antiarmor mine, a direct fire antihelicopter mine, and an indirect fire or sublet-launching antihelicopter mine. The command and control functions that can be performed are turning minefields or individual mines on or off during an exercise.

Like the SAF user interface, smsgui displays a two-dimensional overhead view of the terrain database used in an exercise. In addition it displays outlines of minefields and individual mines within the minefields. Most minefield and mine information is transmitted using the newly defined SMS PDUs. These PDUs are not known to other simulators, so they do not now display mines. The minefield perimeter is transmitted using a previously defined PDU, but apparently no current simulator uses this information to display a minefield.

The SMS user interface program is based on the SAF version 4.3.3 user interface program.[6] The major differences between the two programs are:

- The SAF code to create and display vehicles is not present in smsgui.

- The smsgui has the ability to emplace mines at specific locations while SAF 4.3.3 can emplace mines only as a density within a minefield — it does not determine actual mine locations.

- The smsgui can create smart mines and conventional mines while the SAF user interface can create only conventional mines.

- The smsgui can read and send newly defined SMS PDUs.

The smsgui is a SIMNET application that, during an exercise, requires its own interface to the SIMNET local network. In general this means that the smsgui and the smshost must be run on two separate machines. However, each can be run without the other for special purposes such as testing.

---

[6] The SAF has the same structure as the SMS, with the user interface a separate program from the simulation, running on a different computer.

2. Executing the User Interface Program

The smsgui is installed and launched using standard UNIX procedures. Once running, the basic procedure for creating minefields consists of:

- Moving and sizing the terrain display to focus on the desired area
- Creating an overlay that will contain the minefield
- Drawing an outline of the minefield, adjusting vertex points as desired
- Adding mines within the minefield
- Sending the minefield data to the smshost.

After a minefield has been created and sent to the smshost, it can be controlled (turned on or off).

The following sections describe how to perform these procedures.

a. Installing and Launching

Appendix A has instructions for installing the smsgui.

The smsgui reads a file named sms.lisp to find the location of the smshost, so this file should be checked to ensure correct communications. The file includes information needed by the part of the smsgui adapted from the SAF and these two lines of text:

(sms_host_site   *sitenumber*)

(sms_host_host   *hostnumber*)

The site and host numbers refer to the Simulation Address of the computer running the host program. These are typically found in a file named **assoc.def** on the host program's computer.

The smsgui is launched with the UNIX command line:

smsgui -terrain *tdbname* **-sim_exercise** *ex#* **[-nonet]**

- *tdbname* is the name of the compact terrain database, and should be specified without the full path and without a file extension. The SMS can read only compact terrain databases, not the older-style databases. The full path name constructed by the SMS for the terrain database is **/usr/saf/terrain/***tdbname* **/***tdbname* **.ctdb**.
- *ex#* is the exercise ID for this run, 1<= *ex#* <= 255.

- **-nonet** is an optional switch that specifies stand-alone operation. This can be used to test or practice with the smsgui, or to create minefield overlays in one session and save them for later use in a subsequent session.

After launching **smsgui**, there will be a pause, then a two-dimensional plan view of the terrain database will be displayed in the middle of the screen, with a menu bar across the top and two icons along the upper right side of the screen. (The terrain display and the menu bar operate in the same way as in the SAF, version 4.3.3.)

b. Moving and Sizing the Terrain Display

There are several ways to control the scale of the map display and the area shown. First, the current scale is shown in a box in the lower left of the screen, which serves as a popup menu to select the desired scale. The scale can also be changed using either the middle or right mouse button. To enlarge the display (reduce the scale factor), the middle button can be clicked on the location desired for the display center, or the middle button can be held down while framing a rectangle to view. To reduce the display (increase the scale factor), the right mouse button can be clicked on the desired display center. There are also scroll slides at the right and bottom of the map to move the area shown without changing the scale.

c. Creating or Loading an Overlay

Minefield data are kept in a data structure called an overlay (not the underlying terrain database). There are no fixed limits on the number of overlays that can be created, or the number of minefields in each overlay. Each overlay can be saved to a disk file and subsequently retrieved.

To create an overlay, select the command **Create** from the **Overlay** menu in the menu bar. A small window labeled **Overlay** will appear. Enter a name and if desired a color for the overlay, and when finished select **OK**.

An overlay can be saved using the **Save** command from the **Overlay** menu, and a previously saved overlay can be loaded by choosing the **Load** command and selecting the saved overlay's file name from the list that appears in the **Load Overlay** window.

The current smsgui has some bugs when dealing with overlays and minefields within overlays, and it is recommended that the procedures in this guide be followed closely. It is *not* recommended that an overlay loaded from disk be altered (e.g., by adding

additional minefields, or copying minefields) or that an overlay be altered after creating or loading another on top of it.

d. Creating a Minefield

After creating an overlay, one or more minefields can be placed in it. There are two icon buttons on the upper right of the screen, the top one for placing mines individually within a minefield and the bottom one for creating the minefield itself. When this bottom button is selected a **Minefield** window appears and these steps should be followed:

- **Name the minefield.**

  Select the blank box near the bottom of the window and type a name, followed by <return>. The name will be truncated to eight characters.

- **Draw the minefield.**

  Use the mouse to click (using the left button) on the desired vertex points of the minefield. Up to 13 points can be entered. After entering, vertex points can be dragged to new locations.

The minefield is not complete until mines have been added, so the **OK** button should not be pressed yet.

e. Adding Mines

Mines can be added individually using the mouse to specify each location, or placed uniformly within the minefield by the computer. Both methods may be used within the same minefield. The smsgui will only send to the smshost a maximum of fourteen mines in each minefield. The user must be aware of this limitation, as the smsgui will allow any number to be emplaced. (Only the first fourteen mines will be simulated.)

Individual Mine Emplacement

The procedure for placing mines individually within a minefield is:

- **Select individual emplacement.**

  Select the top icon from the upper right side of the screen. This creates the **Minefield Components** window (the **Minefield** window will disappear).

- **Select a mine type.**

  In the **Minefield Components** window, select the shaded button by the mine type field and drag the cursor over the desired type. All mines you add to

the field will be of this type, until you select a new mine type by repeating this step.

- **Place the mine.**

  Click with the left mouse button at the desired location. The location can be corrected by clicking again at another location, or dragging to another location.

These steps should be repeated until all mines have been placed, then the **OK** button in the **Minefield Components** pressed.

Uniform Mine Emplacement

The controls for uniform mine emplacement are in the **Minefield** window: the individual placement button and the **Minefield Components** window are not used. The procedure for placing mines uniformly within a minefield is:

- **Choose a mine type.**

  The mine types are in a scrollable list in the center of the **Minefield** window.

- **Choose a spacing.**

  The distance between mines is set by sliding the control bar underneath the list of mine types.

- **Display the mines.**

  Select the **Show Changes** button in the **Minefield** window. There will be a brief delay before the display is updated.

- **Adjust individual mine locations.**

  If desired, the mouse can be used to select individual mines and drag them to new locations.

f. Sending Minefield Data

This step should be performed even if running with the -nonet option, in which case a PDU is created but not sent. The procedure is:

- **Select the minefield.**

  If a minefield's **Minefield** window is visible, it is already selected. If not, clicking on the name of a minefield (shown next to the first vertex of the minefield) will select it and cause its **Minefield** window to be displayed.

- **Create and send a new emplacement PDU to the smshost.**

  Select the **OK** button on the **Minefield** window. A dialog box will appear asking if you are really through. If not, select **CANCEL**, otherwise, select

OK. Once **OK** has been selected, no changes can be made to the minefield or to the mines within it (other than turning them on or off).

g. Controlling a minefield

After the emplacement PDU has been sent, the entire minefield or individual mines can be turned off or on (they always start on). To do this:

- **Select a mine or minefield.**

  A mine is selected by moving the cursor over the mine and clicking the left button, while a minefield is selected by moving the cursor over its name and clicking. This causes the **Command and Control** window to appear.

- **Set the state.**

  The state is controlled by the **Activate** button. When the button is depressed (it is shaded to appear below the window surface) it is ON, and when it is not depressed (it is drawn to appear above the window surface) it is OFF.

When the state is set correctly, click the **OK** button. If the state is different from its previous value, a message will be sent to the smshost.

h. Saving Overlays

Overlays, which include minefield and mine locations (but not ON/OFF state), can be saved at any time after emplacement by selecting the **Save** item from the **Overlay** menu.

i. Quitting

The program is terminated by selecting the **Quit** item from the **File** menu.

C. THE HOST PROGRAM

1. Overview

The smshost program creates minefields as directed by the smsgui, and simulates them during an exercise. It also has a test mode in which it operates autonomously, not communicating with the smsgui or other simulators.

The smshost operates with little direct user interaction. After it is launched it communicates with other simulators and with the smsgui over the network. Thus, for example, commands to turn a minefield off are entered at the smsgui and not at the

smshost. The PDUs used by the host are described in Section D and defined in Appendix B. When an exercise has concluded, typing <Ctrl><C> (pressing the Ctrl and C keys simultaneously) terminates host execution.

In addition to PDUs that are required in a simulated exercise and others that collect data about the status of minefields and mines, the smshost can write detailed information to local files. Most of the following description of how to use the smshost concerns this type of output.

2. Executing the Host Program

a. Installing and Launching

Appendix A has instructions for installing the smshost.

The smshost is launched with the UNIX command line:

smshost [*-command value...*]

where [*-command value...*] represents a sequence of one or more commands to the host. These commands tell the host the terrain database, exercise ID, and optionally other information such as flag settings that control output or the name of an input file from which to read additional commands.

b. Commands

Table 4 is a summary list of the commands available to the smshost. The -terrain and -sim_exercise commands are required, while the others are optional. In general, commands can be repeated with the last one encountered overriding earlier ones. Thus, -terrain *name1* followed by -terrain *name2* will result in the use of the name2 terrain database. However, the -testrun command is not changeable (although the length of the test run can be changed), and the -input, -vehicle, and -minefield commands are cumulative in that all occurrences of these commands are processed.

24

Table 4. Host Input Commands

	Command	Value type	Description
	-terrain	name	use terrain database "name"
	-sim_exercise	number	participate in exercise "number"
	-output	name	write output to file "name"
	-input	name	read additional input from file "name"
	-testrun	number	perform a test run (not connected to network) for "number" seconds
	-fAnalData	on/off	turn on/off all other flags
	-fVehicleData	on/off	turn on/off writing data for vehicles
	-fMinefieldData	on/off	turn on/off writing data for minefield state
	-fSAOData	on/off	turn on/off writing data for SAO activity
	-fSensorData	on/off	turn on/off writing data for sensor activity
	-fControlData	on/off	turn on/off writing data for control activity
	-fMunitionData	on/off	turn on/off writing data for control activity
	-fPDUData	on/off	turn on/off writing data for PDU activity
+	-vehicle	number	read "number" vehicle descriptions
+	-minefield	number	read "number" minefield descriptions
+	-reset		reset host for next run (no value is entered)
+	/	text	comment, written to output
+	•	text	comment, ignored

+ available only from input files

Some commands are available only from input files (not the UNIX command line that launches the smshost), and in general setting inputs from a command file can reduce errors or omissions even for standard runs. Thus, the smshost might often be launched with the command

> **smshost -input** *inputfile*

The available commands are:

-terrain *tdbname*

> The smshost constructs the full path name of the compact terrain database from the name given as the value of the -terrain command. The constructed full path name is:
>
> **/usr/saf/terrain/***tdbname***/***tdbname***.ctdb**

-sim_exercise *number*

> The exercise number, which must be between 1 and 255 inclusive.

-output *name*

> A filename to use for output. Output is directed to this file as soon as the command is processed, so in general this command should precede any

25

command that generates output (e.g., the **-terrain** command writes the name of the terrain database).

-input *name*

When this command is encountered the file is opened and read immediately, before any additional commands in the current input stream (except if a **-reset** command is encountered, as described below). Input files can include **-input** commands, so that input can be nested.

-testrun *number*

The smshost operates in test mode, independently of the network. *Number* is the duration of the run, in seconds. All vehicles and minefields used during the run are specified by following **-vehicle** and **-minefield** commands. The test mode is described more fully in subsection 3 below.

-fAnalData *on/off*

Turns on or off all of the other output flags described below.

-fVehicleData *on/off*

-fMinefieldData *on/off*

-fSAOData *on/off*

-fSensorData *on/off*

-fControlData *on/off*

-fMunitionData *on/off*

-fPDUData *on/off*

These commands turn on or off flags that control smshost output. Output will be written to a file if the **-output** command is used to specify the file, otherwise it will be written to the default output (terminal).

-vehicle *number*

The **-vehicle** command is available only from input files, and only during a test run (after the **-testrun** command is processed). *Number* specifies the number of vehicle descriptors to read, which must immediately follow the command line (but which may be interspersed with comment lines). The format of the vehicle descriptor and other aspects of the test mode are described in subsection 3 below.

-minefield *number*

The **-minefield** command is available only from input files, and only during a test run (after the **-testrun** command is processed). *Number*

specifies the number of minefield descriptors to read, which must immediately follow the command line (but which may be interspersed with comment lines). The format of the minefield descriptor and other aspects of the test mode are described in subsection 3 below.

-reset

The **-reset** command is available only from input files, and only during a test run (after the **-testrun** command is processed). When the reset command is encountered, reading from the file containing the **-reset** stops, the run is made, and then input resumes for an additional run. If input streams are nested only the input from the file containing the reset is stopped, and input from higher level files continues. Thus, for example, if the launch command line is:

smshost -input *name1* **-input** *name2*

and the file name1 contains a reset, processing the command line continues following the **-reset** command and file name2 is read. (Only one reset file is remembered so that if file name2 contains a reset, file name1 is never finished.)

/comment

Echo the line to the output file.

*comment

Ignore the line.

3. The Test Mode

The primary purpose for incorporating a test mode was for software development, but it has been left in the deliverable version so that new installations can test the smshost operation before actual exercises. In the test mode, minefields and vehicles are created from input data (rather than from messages received from the smsgui and external simulators). The test mode is initiated using the **-testrun** input command, and vehicles and minefields are input using the **-vehicle** and **-minefield** commands described above.

The number following the **-vehicle** command specifies the number of vehicle records to read, which must immediately follow the command (but which may include comments on lines starting with / or *). A vehicle record describes a single vehicle, and consists of a line describing the vehicle type and initial position and velocity, and zero or more lines describing changes to the vehicle velocity. The association between the input

27

fields (environment, class, etc.) and the vehicle type is described in the SIMNET protocol specification.[7] The vehicle record has these fields, separated by spaces or tabs:

- vehicle ID
- environment
- class
- country
- series
- model
- function
- force
- duration (life, in seconds)
- location (x, y, and z coordinates in successive fields)
- velocity (x, y, and z components in successive fields)
- the number of motion changes, to follow on successive lines.

Each motion change is controlled by a line with these fields:

- time (in milliseconds)
- velocity (x, y, and z components in successive fields).

The number following the -minefield command specifies the number of minefield records to read, which must immediately follow the command (but which may include comments on lines starting with / or *). A minefield record is a single line describing the minefield, followed by one line for each mine in the minefield. The minefield record has these fields, separated by spaces or tabs:

- name
- minefield ID number
- number of vertex points
- time (not used)
- force
- output channel number (not used)
- number of mines (each defined on a following line)

---

[7] *The SIMNET Network and Protocols*, BBN Systems and Technologies Report 7627, June 1991.

28

- vertex points (x and y coordinates in successive fields, with all points on the same input line).

Each line defining a mine has these fields, separated by spaces or tabs:

- mine type
- identification number
- location (x, y, and z coordinates in successive fields)
- initial state.

D. WORKING WITH SIMNET

This section describes important considerations to be aware of when using the SMS in an exercise.

1. The two executable programs of the SMS must be run on separate machines.

The two programs, smsgui and smshost, are both SIMNET applications that read and send messages on the network, and with the current SMS design each must be a separate network node. This is also true of SAF version 4.3.3, so that if the SMS and the SAF are used in the same exercise four machines will be required: one for each of the two user interface programs and one for each of the two host (or back-end) programs.

The smsgui must know the Simulation Address of the smshost. The sms.lisp file gives this information, as explained in Section B.2.

2. All mines use previously defined burst descriptors.

Burst descriptors are used to define how a fire or burst is displayed, and how a target assesses damage. The burst parameters used in the SMS are defined in the param.cc file, and the current version uses the same descriptor types for all mines (see Appendix C). In any exercise the appropriate burst descriptors to use for mines must be considered, and in general descriptors that are used elsewhere in the exercise should not be used for mines.

3. SAF lethality tables can be modified for burst descriptors used by mines.

The SAF maps burst descriptors into munition categories as specified in the file **damage.map.lisp**, and the tables of $P_{kill/hit}$ for the munition categories against SAF vehicles are in the file **df_damg.lisp**. The $P_{kill/hit}$ tables are indexed by impact part (hull or turret), impact location (front, side, back, or top), angle of incidence (30°, 60°, or 90°), and type of kill (catastrophic, mobility, or firepower), so that there are 72 $P_{kill/hit}$ probabilities for each munition category against the following vehicle types: tank, IFV,

29

infantry, helicopter, and airplane. The SMS specifies the same impact geometry for each hit, currently defined in the param.cc file to be in the hull area, at the back, at a 90° angle of incidence. Therefore there are nine relevant $P_{kill/hit}$ values for each munition category that is used: the conditional probabilities of catastrophic, mobility, and firepower kill against tanks, IFVs, and helicopters hit in the back of the hull at a 90° angle of incidence. For use within IDA, SAF vehicle vulnerability tables have been modified to reflect desired lethality characteristics.

4. New PDUs have been defined and are not understood by other simulators or SAF.

The five new PDUs are summarized below and are discussed in Appendix B. These PDUs are defined in the file p_sms.h.

PDU Type	Purpose
Emplacement	For each minefield emplaced by the operator, one or more of these PDUs describes the minefield perimeter and the location, type, and state of each mine in the field. The maximum number of mines in one emplacement PDU is 14, and in the current SMS only one PDU is sent for each minefield (future versions will allow more mines to be placed in each minefield by sending additional PDUs).
Radio Message	Defines a new state (ON or OFF in this version) for one mine or for all mines in a field.
Minefield Status	Describes the number of mines in the field that are: alive, engaged (a subset of alive), exploded, or dead for reasons other than having exploded. Issued every 5 minutes.
Mine Appearance	Describes the type, emplacement time, and appearance (deployed, exploded or activated) for each mine. Sent every 30 seconds.
Mine Status	Provides information on vehicles tracked and launch decisions. Issued every 5 minutes.

In this version of the SMS, emplacement and radio message PDUs are sent from the user interface program to the host program. The remaining three PDUs are created by the host program at 30-second or 5-minute intervals as described above and are sent to all simulators on the network. The intent is to make these PDUs available to a Logger program so the information can be used in the data analysis phase after an exercise.

Since neither manned simulators nor the SAF understands these PDUs, the mines emplaced with the SMS are invisible to all other simulators and to the SAF, with a possible

30

exception noted below. In addition, the SAF has no behavior for responding to the presence of mines.

The SMS host program does issue an existing PDU describing the minefield density. This is the minefield variant of the simulation PDU. The SIMNET protocol specification[8] states that "a simulator receiving a Mine Field PDU may use the information to depict the mine field." However, at this time there are no simulators that display minefields.

---

[8] *The SIMNET Network and Protocols*, BBN Systems and Technologies Report 7627, June 1991, p 95.

IV. MINE MODELS AND ALGORITHMS

The mine models used in the SMS are intended to capture the general characteristics of the developmental landmine systems, but not the engineering details. The SMS is not intended to model mines in sufficient detail to be a tool for engineering development or assessing single engagement performance. Instead, the philosophy has been to develop mine models consistent with the nature of SIMNET and the intended use of the SMS: assessing the battlefield effects of mines.

A. BASIC OPERATION

1. Stand-Alone Objects

All mines comprise three components: a sensor, a control unit, and a munition. For indirect-fire mines, the munition component launches a sublet that itself contains these same three components. The mines and flying sublets are called Stand-Alone Objects or SAObjects, and are the basic objects simulated in the SMS. This section describes general SAObject characteristics, with detailed variations for each SAObject type described later.

During each time cycle of host operation (every 1/10 second) each SAObject is asked to perform an update. Since not every SAObject needs to operate at this rate, the first thing each object does during an update is to check whether it needs to perform one at this time. If not, it does nothing. If so, the update consists of:

- updating the sensor component, which produces a list of potential targets

- updating the control component, which processes the potential target list and determines whether any action is taken

- updating the munition component, which acts on any decision made by the control component.

All mines (but not sublets) have a fixed life span, intended to represent battery life. This power/life is used up at a constant rate whenever the mine is on.

Some of the logic controlling SAObjects and their components is based upon the state that the object is in, and the state is also output to help the analyst understand what is happening within an SAObject. The possible SAObject states are:

- **dead battery** – the power life of the mine has been used up. This state is used only for mines, not sublets.

- **expended** – the SAObject has exploded or fired its munition.

- **off** – the mine has been turned off. This state is used only for mines, not sublets.

- **active** – the SAObject is on and operating normally.

- **detecting** – the sensor is detecting a vehicle. As discussed in the following section, this does not necessarily mean that a real sensor would be detecting a vehicle. Sensors that instantaneously trigger an attack, such as proximity sensors used by conventional mines, are modeled as tracking nearby vehicles.

- **engaged** – the exact meaning of this state varies by SAObject, but in general it means that the control unit of the object has selected a primary target, or that a potential target is close enough to satisfying the attack criteria that frequent updates are called for.

In addition to these SAObject states, each component of an SAObject has a state. In general the state of the SAObject is determined by the states of its components, except that the dead battery and off states are controlled by the SAObject itself.

When the -fSAOData flag is set, each SAObject writes its state whenever a change occurs.

2. Sensor Components

The primary function of the sensor component is to produce a list of potential targets for the control component. Each entry in the list identifies a vehicle and certain properties associated with it such as its position, velocity, and type.

The list does not necessarily correspond to what an actual mine or sublet sensor would produce. For example, a conventional buried mine does not keep track of targets, but merely explodes when it detects one. Nonetheless the SMS conventional mine model keeps a list of vehicles close to it. (The control component causes the mine object to use a higher update rate when a vehicle is close, so that one does not pass over the mine between updates.)

The criteria used to test for inclusion in the target list vary, but typically there is a range test (for close vehicles) and a type test (for certain vehicle types, e.g., helicopters for AHMs). The list is ordered, again with the ordering determined by the SAObject type, but generally closer vehicles are listed first.

34

The way a sensor classifies vehicle types varies, with sensor classifications defined according to the SAObject requirements or the characteristics of actual sensors. For example, conventional mines do not detect air vehicles, and different types of ground vehicles are typically of different size so that the effective "range" of the mine varies. The categories of vehicle types used for the conventional mine are tank, APC, other ground vehicle, and other (not sensed).

In general, errors associated with SAObject operation are modeled in the sensor component. Not all sensors use all types of errors, but there is a consistent way to apply a type of error for those sensors that use it. All errors are defined once at the time a vehicle is added to the sensor list (rather than being independently determined during each update cycle). The error types include:

- Classification errors:

  Some sensors have a probability of thinking that one type of vehicle (e.g., a tank) is something else (e.g., an APC). In no case is the confusion too great – a tank is never mistaken for a helicopter. If the sensor estimates target range, the range error is larger if there is a classification error.

- Range and velocity errors:

  Sensors that estimate range all have a range error. This is applied as a multiplicative factor (e.g., the estimated range is 1.1 times the true range). The initial value of the multiplier E_r is randomly drawn from a uniform $[1.0, 1.0 + e]$ distribution, where e is the maximum error size. Then with 50% probability it is inverted ($E_r = 1.0 / E_r$). If there is an error in classification, it is multiplied by another error factor.

  Velocity errors are determined similarly. The value of the velocity is not correlated with the range error, but if the range error is inverted so is the velocity error.

- Azimuth errors:

  Sensors that produce target location estimates based on acoustic sensors have azimuth errors. These are drawn from a uniform $[-e, e]$ distribution and added to the actual target azimuth. They are not correlated with other errors.

The possible states of the sensor are:

- **on** – the sensor is on and operating normally, but not detecting vehicles

- **tracking** – the sensor is detecting one or more vehicles.

When the -fSensorData input flag is set, each sensor component writes information whenever significant changes occur. These data consist of sensor state

35

changes, vehicles added to the list, vehicles deleted from the list, and changes in the first vehicle on the list.

3. Control Components

The control component examines the sensor's vehicle list and determines whether to take any action. The nature of the decision process varies widely for the different SAObjects. If it makes a decision to fire, it produces the necessary information (such as fire direction for the indirect fire mines) for the munition component.

The possible states of the control unit are:

- **on** – the controller is on and operating normally, but has not selected a target
- **inhibited** – the controller is inhibited because of one or more friendly vehicles in the area
- **engaged** – the controller has selected a target.

When the **-fControlData** input flag is set, each control component writes information about its processing.

4. Munition Components

The munition component performs the appropriate attack on the target, e.g., launching a sublet using the launch parameters received from the control component.

The possible states of the munition are:

- **on** – the munition is on and operating normally, but not attacking a target
- **hit target** – the munition has hit a target (direct fire weapons only)
- **expended** – the munition has attacked a target.

When the **-fMunitionData** input flag is set, each munition component writes information whenever its state changes.

B. CONVENTIONAL MINE MODEL

The SMS currently has only a single, generic conventional mine model representing a landmine with a proximity sensor. Basically, when a vehicle encounters the mine, it explodes. The tracking behavior of the sensor is required by the software design rather than for conceptual reasons, primarily to allow the state of the mine to be updated less frequently when no vehicles are near.

36

1. Sensor

The sensor classifies vehicles as tanks, APCs, other ground vehicles, or other (not ground). It lists all ground vehicles within a specified detection range, and sorts them in increasing (horizontal) range order. There are no sensor errors.

2. Control

The control component determines if the first (closest) vehicle is within the kill range for that vehicle type. Since both mine and vehicle locations are represented as points, the kill range represents half the width of the vehicle. Thus, a 2-meter kill range for tanks represents a 4-meter effective tank width. The "engaged" state means that a target is in kill range, so that it will always occur in conjunction with the munition and SAObject states changing to expended.

3. Munition

The munition always hits the target, sending appropriate messages.

C. WIDE AREA MINE MODEL

The WAM model fires a sublet at a target, as does the actual the WAM now under development. It has a smart sensor that estimates vehicle range and type, and basically tracks a target and fires to intercept at the closest point of approach (CPA). (The actual WAM is planned to use a more complex firing algorithm designed to optimize the sublet/target geometry.)

1. Sensor

The WAM sensor model is intended to represent a sensor that can follow the loudest vehicle sounds that it hears, and that further can distinguish between the sounds of tracked and untracked vehicles.

The sensor classifies vehicles as tanks, other tracked vehicles, heavy wheeled vehicles, light wheeled vehicles, and other (not ground vehicles). The SIMNET vehicle identification codes are somewhat different and in some cases the WAM sensor examines the vehicle function in making its classification (e.g., an unarmored wheeled vehicle that functions as a rocket launcher is classed as heavy). The classification is subject to errors, but only within the tracked or wheeled vehicle types and not between them. For example, a heavy wheeled vehicle might be classed as a light wheeled vehicle but not as a tank.

37

Only ground vehicles within the detection range for their class are included in the vehicle list, which is sorted according to decreasing "loudness." Scaling factors are used to adjust for loudness differences between the different vehicle classifications (e.g., a light wheeled vehicle is assumed to be as loud as a tank 2.5 times as far away). Range and azimuth errors are applied to the vehicle position, with a larger range error used when an error occurs in the vehicle classification. These range adjustment factors are purely arbitrary (and subject to change), but are included for the purpose of ranking targets. Thus, a light wheeled vehicle will not be ranked higher than a tank unless the tank is more than 2.5 times as far away.

2. Control

The WAM controller looks at the first two listed vehicles (the two loudest), and, if one has an estimated CPA within the maximum weapon range, selects a primary target. The first one is the preferred target, unless it is a light wheeled vehicle that is not within the close-in fire range of the mine, and the second is a heavier vehicle (and has a CPA within the maximum weapon range).

If a primary target is found, the controller computes a time to fire based on the time it will take the target to get to the CPA and the time of flight of the sublet. When the estimated delay to fire is less than the normal update cycle time (1 second), the control state is set to "engaged" and the update interval set to the engagement interval (1/10 second). When the delay to fire is less than the engagement update interval, the controller tells the munition to fire the sublet at the CPA. The normal firing logic is overridden when a vehicle reaches a close-in range representing the minimum effective range of the WAM. In this case, the sublet is fired directly at the vehicle.

3. Munition

The munition fires the sublet in the direction of the CPA.

D. INDIRECT FIRE AHM MODEL

Like the WAM, the indirect fire AHM (AhmInd) has a smart sensor that follows and classifies targets, and fires a sublet at the target.

1. Sensor

The AHM systems under development require very precise vehicle classification, with specific helicopter types (e.g., HIND, COBRA) identified according to their acoustic signatures. The SMS sensor model, however, merely classifies vehicles as friendly helicopters, enemy helicopters, and other (not a helicopter).

There are three types of classification errors: identifying a friendly as an enemy, identifying an enemy as a friendly, and identifying the type of enemy helicopter incorrectly. The reason for including an error for identifying an enemy as the wrong type is that range error estimates are likely to be larger in this case. A similar error for misidentifying friendlies is not needed since they are never fired at. There are also range and azimuth errors, but no independent elevation error. (Note that a range error will result in an incorrect altitude estimate, however.)

The sensor lists all helicopters within the detection range for their class, sorted according to range.

2. Control

The control component looks at the entire sensor list, and if there is a friendly on the list inhibits further processing. If there are no friendlies, it selects as its target the first on the list (i.e., the closest) that has a CPA within its lethal envelope. The lethal envelope is based on the Textron AHM design in which the sublet can be fired at a normal elevation angle (45°) to achieve maximum range, or at a higher elevation angle (80°) for shorter range, higher altitude targets. The result is a cylinder with a conic "hat" over the center.

If a target is selected, launch parameters are found from lookup tables. These tables are based on early Textron data analyzing intercept geometries. Without going into great detail, simply firing the sublet at the CPA will not work since the sublet sensor does not look straight ahead but instead scans a circular pattern around its central look angle. The tables are entered with CPA horizontal range and target velocity, and produce a firing range and lead angle.

An estimate of delay before firing is made, and when this delay becomes less than the normal update interval the control state is set to "engaged" and the update interval is set to the engagement update interval. When the delay becomes less than the engagement update interval, the controller tells the munition to fire. The high launch elevation angle is chosen for close, high altitude CPAs.

39

3. Munition

The munition fires the sublet in the indicated direction.

E. DIRECT FIRE AHM MODEL

The direct fire AHM shoots directly at a target, rather than launching a sublet that in turn fires at the target.

1. Sensor

The sensor is the same as the indirect fire AHM sensor. The actual direct fire AHM under development by Ferranti includes a second sensor (IR) to control firing, but in the SMS the effect of this sensor is included in the control model.

2. Control

The control looks through the entire sensor list, and inhibits fire if there is a friendly. Otherwise, it goes into "engaged" mode if there is a target within engagement range, and tells the munition to fire if there is a target within its kill range and in the line of fire (a 45° elevation angle). The condition of a target being in the line of fire is determined by consecutive target elevation angles crossing the firing angle (e.g., one slightly less than 45° and the next slightly greater).

3. Munition

The munition fires at the target, and either hits or misses based on a hit probability parameter.

F. WAM SUBLET MODEL

The WAM sublet under development is designed to spin rapidly around its vertical axis, and has a seeker that looks down at a fixed (35°) angle from the vertical. When combined with the motion of the sublet, the resulting sensed path on the ground approximates a moving circle with increasing radius as the sublet goes up and decreasing radius as it descends. The total ground area covered approximates a long oval. When the IR seeker sees a target, the sublet fires an explosively formed penetrator.

The SMS WAM sublet model follows a ballistic flight path, neglecting drag.

The update interval for a WAM sublet is always short (1/10 second), and the sublet always receives priority processing to ensure that it will be updated every processing cycle.

1. Sensor

The sublet sensor lists all ground vehicles within a detection range great enough to include all nearby potential targets, in increasing horizontal range order. There are no sensor errors.

2. Control

The control looks through the entire sensor list, determining if any vehicle is in the line of fire. It determines this by examining target elevation angles in consecutive updates, and seeing if they cross the firing angle (35° off straight down). The "engaged" state is entered only when a target is attacked.

3. Munition

The munition fires at the target, and either hits or misses based on a hit probability parameter.

G. AHM SUBLET MODEL

The AHM sublet follows a ballistic flight path, neglecting drag, and fires a fragmentation charge when it sees a helicopter. The actual sublet under development has a sensor that looks at a spot 45° away from its central look direction, which is always horizontal for a normal launch angle and 35° above horizontal for a high altitude launch angle. The sublet spins rapidly, yielding a spiral search pattern that moves along the flight path.

The update interval for an AHM sublet is always short (1/10 second), and the sublet always receives priority processing to ensure that it will be updated every processing cycle.

1. Sensor

The sensor lists all helicopters whose range component along its look direction is greater than zero and within a detection range that is somewhat greater than its effective kill range. There are no sensor errors.

2. Control

The control looks through the entire sensor list, attacking the first vehicle within firing range and in the line of fire. It determines the line of fire by examining the angle between the look direction and the target in consecutive updates, and seeing if they cross the firing angle (45°). The "engaged" state is entered only when a target is attacked.

3. Munition

The munition fires at the target, and either hits or misses based on a hit probability parameter.

APPENDIX A

INSTALLATION AND COMPUTER REQUIREMENTS

APPENDIX A
INSTALLATION AND COMPUTER REQUIREMENTS

A. HARDWARE AND SOFTWARE REQUIREMENTS

The SMS runs at IDA on two SGI workstations, either Indigos or Indigo IIs. Both the smshost and smsgui are distributed as executable files that can be run without modification (or recompiling) on these computers.

Since the graphical user interface program is based on the SGI version of the SAF 4.3.3 user interface, it has similar hardware requirements: an SGI workstation with 16 MB of memory and an 8 bit color display. It is written in C using Motif (X11R5). The host program's minimum memory requirements are unknown; it will easily run on an SGI capable of running the SAF 4.3.3 front end or back end. It does not require a color display. The host program is written in C++ (R 3.01) and C. Both the host and user interface programs use SIMNET 6.6.1 protocols.

Compiling and linking either the smsgui or the smshost requires the SAF 4.3.3 developer's libraries of include files and object files. The specific libraries are found in the make files, **make.smshost** and **make.smsgui**, and are discussed in more detail in Section B, Installation Procedure.

It should be possible to compile, link, and execute the SMS on other UNIX workstations that have a SAF 4.3.3 developer's library, but this has not been tested.

B. INSTALLATION PROCEDURE

The SMS is delivered on tape in a tar file. To install the SMS, you will read the tar file from the tape and untar the file. This will create several subdirectories, shown in Figure A-1. The top level subdirectory, sms, contains only two directories, gui and host. In gui, you will find the source code, include files, make file, and executable for smsgui. In host, you will find the corresponding files for smshost. In addition, host contains four subdirectories of test files. There is one subdirectory for each mine type, with four files in each subdirectory. The test files are explained below in Section C, Testing New Installations. Table A-1 lists the files in each directory.

sms

gui   host

conv   dfahm   slaam   slahm
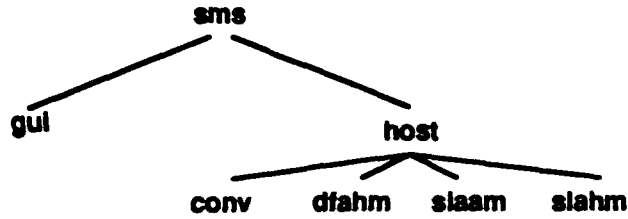
Figure A-1.   SMS Directory Structure

Table A-1.   SMS Directory Contents

directory			
gul			
control.c	main.c	simnet.c	sms_minefield.h
exec.c	make.config	sms.lisp	sms_points.lisp
frames.c	make.smsgui	sms_area.c	smsgui
graphic.c	overlay.c	sms_emplace.c	type_const.h
graphic.h	p_sms.h	sms_minefield.c	xinit.c
host			
assoc.h_mod	cSAObject.h	input	p_sms.h
cControl.cc	cSensor.cc	libctdb.h_mod	param.cc
cControl.h	cSensor.h	list.cc	simman.cc
cMine.cc	cSublet.cc	list.h	simman.h
cMine.h	cSublet.h	main.cc	smshost
cMunition.cc	cTest.cc	make.smshost	type_const.h
cMunition.h	cTest.h	minefield.cc	vehman.cc
cPerformer.h	cTrackfile.cc	minefield.h	vehman.h
cPowerSupply.cc	cTrackfile.h	netman.cc	
cPowerSupply.h	global.cc	netman.h	
cSAObject.cc	global.h	network.h_mod	
host/conv			
ConvMfArray	TestConv	TestConv.out_ida	VehST5
host/dfahm			
DfAhmArray	TestDfahm	TestDfahm.out_ida	VehSH20
host/slaam			
SlaamMfArray	TestSlaam	TestSlaam.out_ida	VehST5
host/slahm			
SlahmArray	TestSlahm	TestSlahm.out_ida	VehSH20

The remainder of this section describes the installation procedure in more detail.

1. Read the contents of the distribution tape. If the tape drive is on another networked computer, use the appropriate form of the command sequence in brackets.

tar xv[f guest@*system* **:/dev/tape] sms.tar**

2. Untar the **sms.tar** file. Since this command will create several subdirectories, make sure you are in the directory from which the new subdirectories should be created.

tar xv sms.tar

3. Optionally, compile both programs. This is not required since the user interface and host programs, smsgui and smshost, are included in the tar file and should at this point already be in your directory.

Before compiling the SMS programs, you should ensure that the path names specified in **/sms/gui/make.smsgui** and **/sms/gui/make.smshost** for include and object files are appropriate for your machine.

If you want to compile the user interface or host programs, you should first copy the existing versions to different files. To save the old and compile a new smsgui, from /sms/gui type:

cp   smsgui smsgui.old

make -f make.smsgui

To save the old and compile a new smshost, from /sms/host type:

cp   smshost smshost.old

make -f make.smshost

4. Copy one or both programs to another workstation, using an appropriate utility such as FTP. The smsgui requires two input files, **sms.lisp** and **sms_points.lisp**, in the same directory it is in. If you move smsgui to another machine, move these files also. In addition the smsgui will read several lisp files from **/usr/saf/config**.

5. Create the link between the user interface and the host program.

During execution, the smsgui will read the **sms.lisp** file and it expects to find the following two lines of text:

(sms_host_site *sitenumber***)**

(sms_host_host *hostnumber***)**

A-3

The site and host numbers refer to the Simulation Address of the computer where the host program will run. These are typically found in a file named **assoc.def** on the host program's computer.

You should now test the programs to ensure that they are working correctly. The smsgui should be tested by creating practice minefields according to the directions given earlier. The next section describes a special testing procedure for the smshost that can be used in addition to practicing with the smsgui.

C. TESTING NEW INSTALLATIONS

As described in Section III.C above, the smshost has a test mode that has been used for developing and testing. Several test files are included in the software distribution, that new installations can use to test the smshost in a stand-alone mode.

There is one test case for each mine type. The files for each test case can be found in the four subdirectories of sms/host: conv for conventional mines, dfahm for direct fire AHMs, slaam for sublet-launching antiarmor mines (WAMs), and slahm for sublet-launching AHMs.

Each test subdirectory contains four files, three input files and one output file for the test case when run at IDA. The specific file names are shown in Table A-2. The three input files contain contain the minefield and mine descriptions, the vehicle descriptions, and commands for executing the test case. Note that the terrain database specified in the command file is for Ft. Hunter-Liggett, which you will need to change if you do not have this database.

Table A-2. Test Case File Names

Mine Type	Minefield File	Vehicle File	Command File	Output File
Conventional	ConvMfArray	VehST5	TestConv	TestConv.out_ida
Direct Fire AHM	DfAhmArray	VehSH20	TestDfahm	TestDfahm.out_ida
Sublet-launching AAM	SlaamMfArray	VehST5	TestSlaam	TestSlaam.out_ida
Sublet-launching AHM	SlahmArray	VehSH20	TestSlahm	TestSlahm.out_ida

To execute a test case you should be in the appropriate test subdirectory. Then, type:

../smshost -input *Testxxxx*

where *Testxxxx* is the the name of the command file. During program execution, the smshost will write progress information to default (screen) output and, in addition, will

create a new output file. The new output file name will be similar to the test output file name shown in Table A-2, without the "_ida" termination. You can now compare the new output file with the one created at IDA. In general there should be some differences attributable primarily to timing variations. Timing differences of 1/10 second are not significant.

APPENDIX B

SMS PDUS

APPENDIX B
SMS PDUS

The SMS PDUs are defined in the C language and C++ language compatible header file **p_sms.h**. The PDUs use the same structure and many of the same data elements as SIMNET PDUs as defined in the SIMNET protocol specification.[9] The purpose of this appendix is to give a verbal description of the fields used in the SMS PDUs. The specific values for some constants used in the PDUs (e.g., SAObject state) are defined in the file **type_const.h**. Excepts from these files are shown in this type face: Courier.

A. TYPE CONSTANTS

```
/* start numbering of SAO types with zero and continue more or less
 consecutively; these types are used to index an array in cMinefield
*/
enum TSAObject {
    SAObjectTypeIrrelevant     = 0,
    kSAOTypeConv               = 1,
    kSAOTypeAhmInd             = 2,
    kSAOTypeAhmDir             = 3,
    kSAOTypeWAM                = 4,
    kSAOTypeSASensor           = 5,
    kSAOTypeACU                = 6,
    kSAOTypeRelay              = 7,
    kSAOTypeWAMSublet          = 8,
    kSAOTypeAhmSublet          = 9,

    maxSAObjectTypes           = kSAOTypeAhmSublet
};

/* start numbering of SAO states with zero and continue more or less
 consecutively; these states are used to index an array in cMinefield
*/
enum ESAOState {
        /* the ordering is important for tests of working/sensing */
    esaosFailed      = 1,
    esaosDeadBat     = 2,
    esaosSelfDest    = 3,
    esaosExpended    = 4,
    esaosOff         = 5,
    esaosLow         = 6,    /*low duty cycle, not now used */
    esaosActive      = 7,    /* regular duty cycle eg 1 sec for WAM
*/
```

---

[9] *The SIMNET Network and Protocols*, BBN Systems and Technologies Report 7627, June 1991.

```
    esaosDetecting      = 8,
    esaosEngaged        = 9,    /*high duty cycle, eg 1/10 sec */

    esaosLowestWorking  = esaosLow,

    maxSAObjectStates   = esaosEngaged
};

enum TMineAppearance { Deployed = 0, Exploded, Activated };
        /*    mine appearance activated used when AHMDirect is spinning
        or Indirect is erect */

enum TRadioMsg        { SmsCommandMsg = 1, SmsStateMsg, SmsSensorMsg };

enum TSmsCommand      { saocmdOn = 1,
                          saocmdOff,
                          saocmdSelfDestruct,
                          saocmdFire };

enum TCommandMode     { CommandModeIrrelevant = 0,
                          Remote = 1, Autonomous = 2,      /* meaning?*/
                          HoldFire = 10, WeaponsFree = 11, /* mines   */
                          Report = 20, Quiet = 21 };       /* sensors */
```

B. THE SMS PDU FAMILY

All SMS PDUs are variants of the basic PDU structure, defined consistently with the SIMNET simulation and data collection PDUs as:

```
typedef struct SmsPDU
{
    SmsProtocolVersion version;
    SmsPDUKind         kind;
    ExerciseID         exercise;
        /* padding */
    unsigned char      unused_1;
    unsigned long      unused_2;
    union
    {
        MinefieldEmplaceVariant     minefieldEmplace;
        MinefieldSAObjectVariant    minefieldSAObject;
        MinefieldRadioMsgVariant    minefieldRadioMsg;
        MinefieldStatusVariant      minefieldStatus;
        SAObjectStatusVariant       SAObjectStatus;
    } variant;
} SmsPDU;
```

Table B-1. SMS PDU Fields

field	definition
version	SMS protocol version number. Version 1 (smsProtocolVersionCurrent = 1) is described in this document.
kind	The PDU variant. Each variant is descibed in a separate section below.
exercise	The exercise number the SMS is participating in.

C. MINEFIELD EMPLACEMENT PDUS

The minefield emplacement PDU defines overall minefield characteristics such as its location, and defines up to 14 SAObjects in the minefield.

```
enum { maxSAOs = 14 };

typedef struct  EmplaceSAObject
{
    TSAObject           SAOType;
    ObjectID            SAONumber;
    WorldCoordinates    SAOLocation;
    ESAOState           initialState;   /* limited to ACTIVE only */
} EmplaceSAObject;

typedef struct MinefieldEmplaceVariant
{
    char                fieldName[8] ;
    ObjectID            fieldID;
    short               nVertices;      /* number of vertices used */
    XYCoordinates       perimeterVertices[maxMinefieldVertices];
    TTime               emplacementTime;
    ForceID             force;
    ObjectID            controlMCU;     /* concept not yet implemented
                                           though field is present here
and
                                           in cMinefield.
                                           Future: minefield will report
                                           status back to MCU. */
    short               nSAOs;          /* number of SAOs present in PDU
*/
    EmplaceSAObject     SAO [maxSAOs];
} MinefieldEmplaceVariant;
```

Table B-2   Emplacement PDU Fields

field	definition
fieldName	Name the user gives to the minefield (up to 8 characters).
fieldID	ID number assigned by the SMS.
nVertices	The number of vertices describing the minefield perimeter, up to 13.
perimeterVertices	Array of perimeter vertices.
emplacementTime	Time the smshost will emplace and activate the minefield. [Not used currently.]
force	Force the mineifield belongs to. [Always distinguished currently.]
controlMCU	Radio channel associated with reporting minefield data. [Not used currently.]
nSAOs	The number of SAObjects in the minefield, up to 14.
SAO [maxSAOs]	Array of EmplaceSAObject structures defining the SAObjects.

field	definition
SAOType	The SAObject type.
SAONumber	ID number assigned by the SMS.
SAOLocation	The SAObject location
initialState	Initial state, always set to Active currently.

D. SAOBJECT PDUS

The SAObject PDUs are counterparts of VAPs. They identify the SAObject and give its location.

```
typedef struct SAMineData
{
    TMineAppearance    appearance;
} SAMineData;

typedef struct MinefieldSAObjectVariant
{
        /* identity */
    ObjectID        SAObjectID;
    ForceID         force;
    TSAObject       objectType;

    /* appearance */
    XYCoordinates  location;      /* do we really need WorldCoords? */
    TTime          emplacementTime;
    long           padding;  /* placeholder . */
    union
    {
        SAMineData     mineData;
    } SAObjectData;

} MinefieldSAObjectVariant;
```

Table B-4   SAObject PDU Fields

field	definition
SAObjectID	ID number assigned by the SMS.
force	Force the SAObject belongs to. [Always distinguished currently.]
objectType	The SAObject type.
location	Location.
emplacementTime	Time the SAObject (and minefield) was created.
mineData	Mine appearance [deployed, exploded, or activated.]

E. RADIO MESSAGE PDUS

The only radio message that is currently implemented is an on/off command.

```
typedef struct CommandRadioMsg
{
    TTime               effectiveTime;   /* for the action to occur */
                                         /* not used in first version */
    TTime               endTime;         /* if command has a duration */
                                         /* not used in first version */
    TSmsCommand         command;
    TCommandMode        mode;
    unsigned long       unused[2];       /* possible future data field? */
} CommandRadioMsg;

typedef struct MinefieldRadioMsgVariant
{
    ObjectID            sender;
    TRadioChannel       destination;
    TTime               sendTime;
    TRadioMsg           messageType;
    union
    {
        CommandRadioMsg     command;
        StateRadioMsg       state;    /* not yet implemented */
        SensorRadioMsg      sense;    /* not yet implemented */
    } radioMsgData;
} MinefieldRadioMsgVariant;
```

Table B-5   Radio Message PDU Fields

field	definition
sender	Sending object ID.
destination	Destination object ID.
sendTime	Time the command is sent.
messageType	Radio message type. [Only command messages currently implemented.]
command	Command data.

Table B-6   Command Structure Fields

field	definition
effectiveTime	Effective time. [Always now in current version.]
endTime	Duration of command action. [Not used in current version.]
command	Command type. [Only on/off in current version.]
mode	Not used in current version.

F. MINEFIELD STATUS PDUS

The minefield status PDUs are designed to provide data for later analyses.

```
typedef struct SAObjectCount
{
    TSAObject  SAObjectType;
    short      numAlive;
    short      numExploded;
    short      numDead;
    short      numEngaged;
} SAObjectCount;

typedef struct MinefieldStatusVariant
{
    ForceID        force;
    ObjectID       minefieldID;
    char           fieldName[8];
    SAObjectCount  SAOsummary[maxSAObjectTypes];
    long           unused_4[7];
} MinefieldStatusVariant;
```

Table B-7   Minefield Status PDU Fields

field	definition
force	Force the minefield belongs to. [Always distinguished in current version.]
minefieldID	Minefield ID number.
fieldName	Minefield name.
SAOsummary	Array giving the number of SAObject of each type, indexed by state.

Table B-8   SAObject Count Structure Fields

field	definition
numAlive	The number of objects of this type that are alive.
numExploded	The number of objects of this type that are exploded.
numDead	The number of objects of this type that are dead.
numEngaged	The number of objects of this type that are engaged.

G. SAOBJECT STATUS PDUS

The SAObject status PDUs are designed to provide data for later analyses.

```
typedef struct MineData
{
    ESAOState  currentState;

    long       errors;            /*not used in this version */
    short      vehiclesCurrent;   /*currently tracked or engaged */
    short      vehiclesTracked;    /*total over time*/
                                  /* not used in this version */
    VehicleID  vehicleAttacked;   /*whether current or previous */
```

```
        TTime              attackTime;
        long               attackRangeEst;    /* estimated at time of attack */
        long               attackRangeError;
        long               attackAzEst;       /* estimated at time of attack */
        long               attackAzError;
        long               attackCPA;         /* estimated at time of attack */
        FireResult         attackResult;   /* ground, vehicle, or nonimpact */
} MineData;

typedef struct SAObjectStatusVariant
{
    ForceID        force;
    ObjectID       SAObjectID;
    TSAObject      SAObjectType;
/* doublecheck amount of padding needed after other data types  */
/* are added to the union of specificData */
    long           unused_3;
    union
    {
        MineData    mineData;
/*         SASensorData    sensorData;  */
/*         SAControlData   controlData; */
/*         SARelayData     relayData;   */
    } specificData;
} SAObjectStatusVariant;
```

Table B-9   SAObject Status PDU Fields

field	definition
force	Force the SAObject belongs to. [Always distinguished in current version.]
SAObjectID	SAObject ID number.
SAObjectType	SAObject type.
mineData	Sturcture describing mine status.

Table B-10   MineData Structure Fields

field	definition
currentState	The current state of the mine.
errors	Placeholder, not used in current version.
vehiclesCurrent	The number of vehicles being tracked.
vehiclesTracked	Total number of vehicles that have been tracked. [Not used in current version.]
vehicleAttacked	The ID of the vehicle that was attacked, if any.
attackTime	The time of attack.
attackRangeEst	The estimated range of the attacked vehicle.
attackRangeError	The range error value for the attacked vehicle.
attackAzEst	The estimate azimuth of the attacked vehicle.
attackAzError	The azimuth error value for the attacked vehicle.
attackCPA	The estimate CPA range for the attacked vehicle.
attackResult	The result of the attack.

APPENDIX C

PARAMETER SUMMARY

APPENDIX C
PARAMETER SUMMARY

These parameters are defined in the file param.cc.

Conventional mine:

component	parameter	index	value
mine	regular update interval		1 second
	frequent update interval		1/10 second
	lifespan		24 hours
sensor	detection range		30 meters
control	kill range	tank	2 meters
		APC	1.5 meters
		other ground	1 meter
munition	burst type	projectile	105APDS
		detonator	M739 155mm

WAM:

component	parameter	index	value
mine	regular update interval		1 second
	frequent update interval		1/10 second
	lifespan		24 hours
sensor	detection range	tank	200 meters
		APC	140 meters
		heavy wheel	100 meters
		light wheel	50 meters
	loudness adjustment factor	tank	1.0
		APC	0.7
		heavy wheel	0.4
		light wheel	0.2
	classification error probability		0.05
	range error multiplier		0.1
	azimuth error factor		0.06
	velocity error multiplier		0.1
control	maximum range		100 meters
	close-in range		15 meters
munition	launch burst type	projectile	TOW
		detonator	M739 155mm

WAM Sublet:

component	parameter	index	value
sublet	regular update interval		1/10 second
	horizontal velocity		28.84 meters/second
	initial vertical velocity		22.84 meters/second
	burst when hitting ground	projectile	TOW
		detonator	M739 155mm
sensor	detection range		50 meters
control	firing angle		-55 degrees
munition	fire burst type	projectile	TOW
		detonator	M739 155mm
	impact burst type	projectile	105APDS
		detonator	M739 155mm
	hit probability		0.7

AHM Indirect Fire Mine:

component	parameter	index	value
mine	regular update interval		1 second
	frequent update interval		1/10 second
	lifespan		24 hours
sensor	detection range		2000 meters
	classification error probability	friend to enemy	0.01
		enemy to friend	0.05
		wrong enemy	0.03
	range error multiplier for class error		0.8
	range error multiplier		0.1
	azimuth error factor		0.06
	velocity error multiplier		0.1
control	maximum horizontal range		180 meters
	cylinder altitude		120 meters
	maximum altitude		180 meters
	max range for high launch		120 meters
	min altitude for high launch		50 meters
munition	burst type	projectile	TOW
		detonator	M739 155mm
	launch angle	regular	45 deg
		high	80 deg

AHM Indirect Fire Sublet:

component	parameter	index	value
sublet	regular update interval		1/10 second
	horizontal velocity	regular launch	26 meters/second
		high angle launch	6.38 meters/second
	initial vertical velocity	regular launch	26 meters/second
		high angle launch	36.26 meters/second
	burst when hitting ground	projectile	TOW
		detonator	M739 155mm
sensor	detection range		200 meters
control	firing angle		45 degrees
	maximum firing range		140 meters
munition	fire burst type	projectile	TOW
		detonator	M739 155mm
	impact burst type	projectile	105APDS
		detonator	M739 155mm
	hit probability		0.8

AHM Direct Fire Mine:

component	parameter	index	value
mine	regular update interval		1 second
	frequent update interval		1/10 second
	lifespan		24 hours
sensor	detection range		2000 meters
	classification error probability	friend to enemy	0.01
		enemy to friend	0.05
		enemy to friend	0.03
	range error multiplier for class error		0.8
	range error multiplier		0.1
	azimuth error factor		0.06
	velocity error multiplier		0.1
control	fire angle		45 degrees
	maximum firing range		100 meters
	engagement range		300 meters
munition	fire burst type	projectile	TOW
		detonator	M739 155mm
	impact burst type	projectile	105APDS
		detonator	M739 155mm
	hit probability		0.8

APPENDIX D

APPROVED DISTRIBUTION LIST FOR IDA DOCUMENT D-1452

APPENDIX D
APPROVED DISTRIBUTION LIST FOR
IDA DOCUMENT D-1452

Department of Defense   **Number of Copies**

Department of Defense
OUSD(A)/TWP/LS
Room 3D139, the Pentagon
Washington, DC 20301-3071
    ATTN: Mr. Andrus Viilu
          Room 3B1060   1

Advanced Research Projects Agency
Correspondence Control, 8th Floor
3701 N. Fairfax Drive
Arlington, VA 22203-1714
    ATTN: T. Hafer   1
          J. Wargo   1
          D. McBride   1
          R. Murphy   1
          A. Coonce   1

Defense Technical Information Center
Cameron Station
Alexandria, VA 22304-6145   2

Department of the Army

Department of the Army
Armament Research, Development, and Engineering Center
Central Mail Room, Bldg 1
Picatinny Arsenal, NJ 07806-5000
    ATTN: SMCAR-ASH Ostuni B/1   1
              Julie Chu   1
        SMCAR-FSM-S Bldg 94
              Wayne Ahlers   1
              Kevin Wong   1
        SMCAR-FSF-BD Bldg 95
              Tom Peters   3
              Israel Lorenzana   1
              Mike Divincenzo   1

Commanding Officer
Department of the Army
USAADASCH
Fort Bliss, TX 79916
    ATTN: ATSA-CDF, Maj Stephens   1

Commandant
USAES
Fort Leonard Wood, MO 65473-6620
    ATTN: ATSE-CDA
                              David Lowenthal   1
                              Maj LeRoy Maurer   1

Department of the Army
Army Material Command
STRICOM
12350 Research Parkway
Orlando, FL 32826-3276
    ATTN: AMCPM-TND-EC, Gene Wiehagen   1
             AMCPM-CATT, Col Shiflett   1

Commanding Officer
Department of the Army
USA Belvoir RD&E Center
Fort Belvoir, VA 22060-5606
    ATTN: SATBE-N   Pam Jacobs   1
                            Hugh Carr   1

Commandant
Department of the Army
Field Artillery School
Fort Sill, OK 73503-5600
    ATTN: ATSF-CBL   1

Commanding Officer
Department of the Army
USAIS
Fort Benning, GA 31905-5400
    ATTN: ATSH-WC   1

Commander
Department of the Army
HQ TRADOC
Fort Monroe, VA 23651-5000
    ATTN: ATCD-L   1
             ATCG-S, Dr. Paul Berenson   1

Commandant
Department of the Army
USAARMC
Fort Knox, KY 40121-5000
    ATTN: ATZK-MW   1

Director
Department of the Army
System Analysis Activity
Aberdeen Proving Ground, MD 21005
    ATTN: Will Brooks   1

Commander
Department of the Army
MICOM
RFPI PRDJ Office
Redstone Arsenal, AL 35809
    ATTN: AMSMI-RD-AS-IR BLDG 5400, Ms. Emily Vandiver   1

Other Organizations

Loral Training and Technical Services
2021 Blackhorse Regiment
Fort Knox, KY 40121
    ATTN: Paul Monday   1

Loral Advanced Distributed Simulation
50 Moulton St.
Cambridge, MA 02138
    ATTN: Dr. Andy Ceranowicz   1

Institute for Defense Analyses   30
1801 N. Beauregard Street
Alexandria, VA 22311-1772