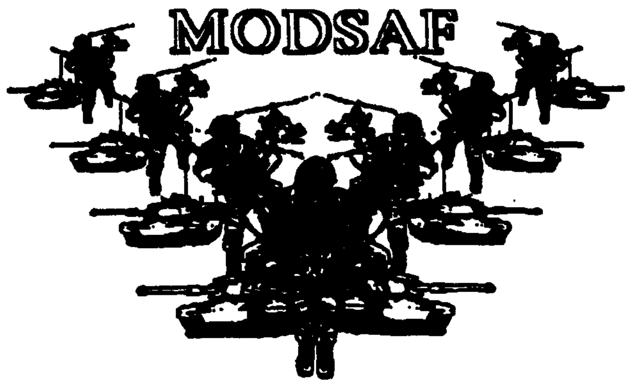
1

LORAL
Systems Company

ADST



Advanced Distributed Simulation Technology

Modular Semi-Automated Forces System (MODSAF)
Maintenance Plan
CDRL A00A

LORAL
Systems Company
ADST Program Office
12443 Research Parkway, Suite 303
Orlando, FL 32826

February 19, 1993
Prepared for
STRICOM
Simulator Training and
Instrumentation Command
Simulator Training and Instrumentation Command
Naval Training Systems Center
12350 Research Parkway
Orlando, FL 32826-3275

\5pf

LORAL
Systems Company

REPORT DOCUMENTATION PAGE

Form approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 02/15/93	3. REPORT TYPE AND DATES COVERED Version 1.0 12/92 - 02/93

4. TITLE AND SUBTITLE MODSAF Maintenance Plan	5. FUNDING NUMBERS Contract No. N61339-91-D-0001 CDRL A00A
6. AUTHOR(S) Ceranowicz, Andrew	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Loral Systems Company ADST Program Office 12443 Research Parkway, Suite 303 Orlando, FL 32826 — BBN Advanced Simulations BBN Systems and Technologies Division Bolt, Beranek, and Newman Inc. 50 Moulton Street Cambridge, MA 02138	8. PERFORMING ORGANIZATION REPORT NUMBER ADST/WDL/TR-92-003050 CRDL A00A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) STRICOM Naval Training Systems Center 12350 Research Parkway OrlandoFL 32826-3275	10. SPONSORING ORGANIZATION REPORT
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE A

13. ABSTRACT (Maximum 200 words)

The MODSAF Maintenance Plan describes the process required to modify and change parameters, features, characteristics, and behaviors of the software modules of Version 1.0 of the Modular Semi-Automated Forces System (MODSAF) 1.0. MODSAF is a Distributed Interactive Simulation (DIS) system for simulating and controlling entities such as vehicles, dismounted infantry, missiles, and dynamic structures on a virtual battlefield. These entities interact with each other and manned simulators to support training, combat development experiments, test, and evaluation studies in the DIS environment. MODSAF is the replacement for the current SIMNET Semi-Automated Forces (SAF).

14. SUBJECT TERMS			15. NUMBER OF PAGES 10
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	17. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	17. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std Z39-18
298-102

Table of Contents

0.1 Introduction

The ModSAF software architecture is an extensible set of software modules that allows rapid development and testing of new agents and tactics in the DIS simulated environment. The platforms, sensors, weapons, organizations, and tactics in use by the military are constantly changing. In addition, the scope of systems simulated by ModSAF 1.0 is limited. Therefore, extensions and modifications will be needed to support future experiments and training. The architecture of ModSAF supports this process by making it easier to add new modules. Therefore, it is anticipated that the users of ModSAF will wish to extend and customize it. This plan indicates the types of changes that users can make and describes some of the elements involved. Since ModSAF is currently under development, it is not feasible to provide the exact details of the procedures required to make the changes in this document. ModSAF will include detailed online documentation describing these procedures.

ModSAF can be changed through parameter files or by adding new code modules. New entities can be added, behaviors can be modified, and the user interface can be changed by the use of parameter and resource files. More fundamental changes require writing new code. ModSAF supports the addition of new code by isolating the code simulating different components and behaviors in libraries. These libraries have well defined and documented interfaces and explicit dependencies. This allows the implementor of a change to quickly determine what libraries need to be added or modified.

0.2 Adding and Changing Platforms

The ModSAF architecture only defines a generic simulation entity in code. Detailed specification of entities is actually done at when the entity is created at the start of an exercise. The components and behaviors of the entity are read from a database of entity descriptions defined in parameter files or by user inputs. The parameters specify the characteristics and behavior of the entity and allow these to be changed without recompiling the system.

All the simulation modules which make up a particular SAFOR entity are listed in a data file stored with the main program. This allows the user to change, add, or delete components such as sensors, weapons, and propulsion systems. A generic version of each component is defined in a library. If you write your own weapon or sensor component you can include it in a vehicle by modifying the components section of the vehicle's parameter file. Generic components (such as tracked-hull) are parameterized so they can be customized for particular vehicles (M1, M2, T72, T80, etc.). This is true not only of physical components, but also of behavioral descriptions and architectural support modules. For each component included in the entity, the data file also lists the parameters used for that entity.

Parameter files are distinguished by ".rdr" extensions on their names. For example, the parameter file defining an F14 aircraft model is named "*US_F14D_params.rdr*". The F14 parameter file specifies the dynamics model to use (rotary wing aircraft, fixed wing aircraft, tracked ground vehicle, etc.), and the parameters of that model (maximum turn rate, fuel consumption, etc.). The network appearance of the entity and its dead reckoning thresholding parameters are specified. Weapons systems are specified by name, and vehicle dependent weapons parameters are given for each (time to load the weapons, accuracy as a function of range, etc.). Since all this data (and much more) is specified for each vehicle, it is possible to change the performance characteristics of each vehicle without modifying any software (as long as the basic component models are capable of the desired behavior). Adding new kinds of vehicles often requires only copying and modification of data files.

All parameter files can be edited with standard Unix text editors such as vi or emacs. Future versions of ModSAF will support specialized editors with graphical interfaces, menus, and error checking features.

Three different types of files may be modified or added when changing or adding a platform to ModSAF.

- A *model file* may be modified to change the parameters for a particular type of platform. This model file might define parameters that are shared by different platforms. A new model file might be created to specify parameters for a new type of platform.
- The *mapping file* may be modified to specify new mappings between DIS entity types (platforms) and the parameters used to simulate these platforms.
- The *model list file* may be modified to specify the loading of newly defined model files.

These files are described below.

0.2.1 Model List File

At program startup, the file "modellist.rdr" is read by the ModSAF application program. This file contains the list of all the vehicle model files that should be read in by the program, as well as some utility files that define macros which are shared between multiple vehicle model files. The following is an example modellist file:

```
 (
  ;; Generic Macro files
  "soils.rdr"             ; Defines symbolic soil types

  ;; Specific Vehicle parameter files
  "US_F14D_params.rdr"
  "US_Sidewinder_params.rdr"
  "US_Phoenix_params.rdr"
  "US_Sparrow_params.rdr"

  "US_M1_params.rdr"
  "USSR_T72M_params.rdr"

  ;; Add more vehicles here

  ;; Finally, the association between vehicles and model data
  "models.rdr"
 )
```

This file specifies the loading of the following files:

1. "soils.rdr" which contains definitions of various soil types used by certain model files
2. Model parameter files for the following platforms:
   - F14D fixed wing aircraft
   - Sidewinder, Phoenix and Sparrow air-to-air missiles
   - M1 and T72M main battle tanks
3. A mapping file ("models.rdr") which specifies what parameters should apply to what DIS entity types.

0.2.2 Model File

A model file specifies all the parameters for a particular platform. Parameters are specified in a macro format that allow different vehicles to share common parameters. Each model file defines one macro which defines all the relevant parameters that are applicable for a vehicle. This macro can then be referenced in the model mapping file to specify what DIS platforms use these parameters.

The following is an excerpt from a model file for the F14D aircraft. It defines a macro US_F14D_MODEL_PARAMETERS which specifies various types of parameters for the simulation of that model.

```
US_F14D_MODEL_PARAMETERS
(SM_PBTab)
(SM_TaskManager)
(SM_Entity (length_threshold 10.0)
           (width_threshold 10.0)
           (height_threshold 10.0)
           (rotation_threshold 3.0)
           (turret_threshold 3.0)
           (gun_threshold 3.0)
           (vehicle_class vehicleClassSimple)
           (guises vehicle_US_F14D vehicle_USSR_Su25)
           )
(SM_Collision (check ground trees missiles platforms)
              (announce ground)
              (duration 5000)
              (feature_mass 10000)
              (fidelity high))
(SM_DFDamage (filename "dfdam_vulnerable.rdr")
              (damage_threshold 10.0))

(SM_Components (hull SM_FWAHull)
               (station-1a [SM_MissileLauncher | 0])
               (station-3  [SM_MissileLauncher | 1])
               (station-5  [SM_MissileLauncher | 2])
               (apg-71     [SM_Radar | 0]))

(SM_FireControl (components station-1a station-3 station-5))

(SM_FWAHull
 (c_drag_super 1.06)
 (c_drag_sub 1.0)
 (vehicle_mass 27000.0) ;; Kg
 (thrust_min 0.0)       ;; Newtons
 (thrust_max 308000.20) ;; N
 (lift_min  -529200.0)  ;; N, corresponds to -2G's structure limit on F14D
 (lift_max  1852200.0)  ;; N, corresponds to +7G's structure limit on F14D

...
```

This F14 model includes a task manager to execute its behaviors, an entity component to keep track of data required for the DIS entity state packet and determine when network packets should be output, a collision detection component, a damage evaluation component, a hull, 3 missile launchers, a radar, and a fire control system.

0.2.3 Mapping File

A mapping file specifies the mapping between DIS entity types and the parameters used to simulate such entities. For example, there are different DIS entities representing variations on the F14 aircraft, such as F-14A, F-14B, F-14C, etc. ModSAF allows the simulation of each of these different aircraft using the same parameters specified by the macro defined in the US_F14_params.rdr file.

The name of the mapping file to use specified as the last file listed in the model list file "modellist.rdr".

The following is an example mapping file ("models.rdr"):

```
;; $Revision: 1.2 $

(
 ("vehicle_US_F14D"          US_F14D_MODEL_PARAMETERS)
 ("vehicle_US_F14D-Soar"     (SM_SAFSOAR) US_F14D_MODEL_PARAMETERS)
 ("munition_US_Sidewinder"   US_SIDEWINDER_MODEL_PARAMETERS)
 ("munition_US_Phoenix"      US_PHOENIX_MODEL_PARAMETERS)
 ("munition_US_Sparrow"      US_SPARROW_MODEL_PARAMETERS)

 ("vehicle_US_M1"     US_M1_MODEL_PARAMETERS)
 ("vehicle_USSR_T72M" USSR_T72M_MODEL_PARAMETERS)
)
```

This file specifies:

1. The DIS entity `vehicle_US_F14D` should use the parameters defined by the macro called `US_F14D_MODEL_PARAMETERS`. This macro happens to be defined by the model file '`US_F14D_params.rdr`'.
2. The DIS entity `vehicle_US_F14D-Soar` should use a parameter called `(SM_SAFSOAR)`, as well the other parameters specified in the `US_F14D_MODEL_PARAMETERS`.
3. The Sidewinder, Phoenix, and Sparrow missile entities should use the parameters specified by the respective macros.
4. The M1 and T72M tank entities should use the parameters specified by the respective macros.

0.3 Adding and Changing Units

Units are represented by ModSAF as dynamic structures in the Persistent Object Database. These unit structures can be edited by the SAF Commander during an exercise to perform dynamic task organization. Additionally certain tasks and missions will automatically perform task organization during the exercise to achieve their objectives.

The definition of the basic units available to the system is done through parameter files that specify the subordinate units, applicable formations, capabilities, and applicable missions.

0.4 Adding and Changing Physical Components

Each vehicle or entity simulated by ModSAF is composed of a number of components. These component models are defined by libraries and are parameterized so that they can be used to represent a range of systems. If the desired system can be represented by an existing component then you can create the appropriate model by editing parameter files. If there is no applicable model you must write your own component library.

Components are classified into the following categories:

Hulls	These are physical models of vehicle hull capabilities. A particular vehicle never has more than one hull. Examples include libTrackedHull, libFWAHull, and libMissileHull.
Turrets	These are physical models of turret capabilities. A vehicle may have more than one turret. Only one kind of turret (libGenTurret, for generic turret) is currently defined.
Guns	These model various types of munition launchers. Examples include libBallGun, and libMissileLauncher.
Sensors	These are the parts of a vehicle which are responsible for determining what may be sensed by the vehicle crew. Examples include libVisual, LibRadar, and in the future, libAural (for infantry).

The collection of physical subsystems, some of which may occur more than once on a particular vehicle (such as a vehicle with two ballistic guns) is managed by libComponents.

To support interoperability and module replacement, the following conventions are used in ModSAF to guarantee the protection of private data:

- It is recognized that any public data structure may at least be examined by software in higher levels, and hence the meaning of values should be well documented.
- The modification of public data structures is strictly prohibited, except by way of a function invocation.

- All public header files (where data types and global variables are specified) defined by a library are copied to a public area at compile time, while private header files are never copied from the source area.
- Private functions are defined as 'static' wherever possible (this strictly prevents the compiler from allowing other modules to call them).

The following are special guidelines that must be followed when defining vehicle sub-class libraries:

— The library is built using '`osatemplate`' (the ModSAF templating program), to ensure that all necessary functions have been provided.

— Where applicable, the first argument to public functions is a vehicle ID.

— No public function assumes that the class exists for the vehicle ID specified. Each is prepared to return a nominal value, in case one is not specified correctly. In general, error messages are not printed when this happens.

0.5 Changing Behavior Via Task Frames

The behavior of ModSAF vehicles and units is defined by tasks and task frames. A *task* is a behavior, such as obstacle avoidance or targeting, defined by parameterized library. *Task frames* are used to group related tasks which run at the same time and represent a phase of a mission such as road march. The task frames that the user is presented with are similar to the *Combat Instruction Sets* in previous SAF systems. These task frames are defined in data files that list the tasks included in each task frame together with the parameters for those tasks. Parameters may be defaults that the user may edit or system-level parameters that the user may not change. The parameters all task frames to customize the operation of the tasks which it uses.

0.6 Changing Behavior Via Tasks

The foundation of the ModSAF command and control framework is the task. Tasks may control a unit (company road march) or they may control an individual entity (drive toward a waypoint). Tasks may be executed to achieve a mission objective (attack an objective), or they may be executed continuously, independent of the mission (scan for enemy). Tasks may be representations of actual battlefield behavior (run for cover), or they may be implementation details of the simulation (arbitrate between several possible alternative actions).

Each task is encoded as a finite state machine. A special entity component called the task manager (libTaskMgr) is responsible for triggering the executing the tasks for each entity. A subset of the current state of each executing task is maintained on the network so that the platform executing the task may be seamlessly transferred from one ModSAF simulator to another.

To add a new task you can customize and existing task by changing its parameters. These tasks are called derived tasks and they are defined by parameter files. Each derived task refers to a generic task defined by a ModSAF library and provides a new name and parameters for the task. To define a completely new behavior you will have to create a new library. These tasks can be defined as finite state machines using the ModSAF finite state machine preprocessor or as arbitrary code to implement other paradigms. In addition to the things needed to define a new physical component, some tasks require the definition of a user interface editor that allows the SAF commander to inspect and modify the execution of the task. These editors are specified in data files.

0.7 Making Protocol Changes

The ModSAF system interfaces to the DIS network using SNIP. Any changes of syntax that occur in DIS can be accommodated by changing only the Simulation Protocol Dependent Module of SNIP. SNIP Simulation Information Units isolate ModSAF from changes in the protocol. The addition of new packet types will also require the addition/modification of SAF behaviors and physical components that can react to or generate the new packets.

0.8 Making GUI Changes

The Graphical User Interface is composed of many independent *editors*, around a substructure of support libraries. The architectural support comes from libSAFGUI, which provides the top-level layout, manages the current interface *mode*, and displays help; libSensitive, which allows objects on the map (such as points, or units) to be classified into mouse sensitive groups; libTactMap which provides a 2D view of the terrain database, as well as the ability to add dynamic objects to the map display; libEditor, which provides a means for defining editors in data files; libPrivilege, which allows access to buttons on the display to be restricted; and libXFile, which provides a simple means for graphically interacting with the Unix file system.

A data file specifies the method for drawing various classes of vehicle types by choosing from a small set of graphic primitives (box, line, circle, etc.). Sizes, locations, and rotation (with the hull, or with the turret) information is given for each attribute, as well as the order in which to draw them.

0.9 Creating New Applications

The ModSAF libraries form a repository of software. Subsets of this software can be linked together by a main application program together with parameter files for defining the simulation objects and user interface to create new ModSAF systems or subsystems. The ModSAF main.c program and src/ModSAF directory provides an example of how to create these systems. Typically very little new code is required to create a new ModSAF system from the library components.

0.10 Adding New Databases

The terrain is represented in various formats contained in two terrain databases for use by the simulation modules.

The vector format provided by libCTDB provides high performance algorithms for computing point to point visibility, radar masking, elevation lookup, vehicle placement on the terrain, and graphical display of the terrain database (such as contour line generation and hypsometric mapping).

The object based format provided by libQuad provides a structure more suitable for intelligent terrain processing.

These databases are compiled from BBN's S1000 Source Databases. The addition of new databases will require the creation of a new S1000 database using the BBN S1000 toolkit. Then the source database would need to be processed by the LibQuad and LibCTDB compilers to produce new databases.

0.11 Documentation Requirements

Every ModSAF library must be documented using texinfo format documentation. A template document can be found in '`/usr/local/lib/osa/texinfo`'. The texinfo documentation should be named '`lib<lib>.texinfo`'. It should translate to both emacs info format ('`make info`') and '`.dvi`' format ('`make lib<lib>.dvi`') without errors or warnings.

0.12 Testing Requirements

Each library should contain a test program or test plan which can be used to confirm that the library works correctly. This program should be well commented so that it can be understood by other programmers trying to determine if a port or a change has worked, and so that it can be used as an example.

The test program should have '`test`' in its name (such as '`test.c`', '`xtest.c`', etc.), so that it can be readily identified as a test program.

Test programs which determine their own success or failure are preferable to those which require some analysis of their output by the tester.

0.13 Coding Standards

If new library code is written in C and especially if that new code is added to a currently existing library, the coding standards defined in the ModSAF Programmer's Guide should be followed. This document is part of the ModSAF documentation distribution available both online and in hardcopy.