Navy Personnel Research and Development Center

San Diego, CA 92152-6800 TN 89-18 April 1989

BATMAN (Battle-Management Assessment System) & ROBIN (Raid Originator Bogie Ingress): Rationale, Software Design, and Database Descriptions

NPRDC TN 89-18 April 1989

BATMAN (Battle-Management Assessment System) & ROBIN (Raid Originator Bogie Ingress): Rationale, Software Design, and Database Descriptions

Pat-Anthony Federico
Navy Personnel Research and Development Center

Steven H. Bickel
Randy R. Ullrich
Thomas E. Bridges
Brian Van de Wetering
Systems Engineering Associates

Reviewed and released by
E. G. Aiken
Director, Training Technology

Navy Personnel Research and Development Center
San Diego, California 92152-6800

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A207 223

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NPRDC TN 89-18	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Navy Personnel Research and Development Center 6b OFFICE SYMBOL (If applicable) Code 15	7a. NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and ZIP Code) San Diego, CA 92152-6800	7b. ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING / SPONSORING ORGANIZATION Chief of Naval Operations 8b. OFFICE SYMBOL (If applicable) CNO (OP-11)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code) Washington, DC 20350-2000	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 63720N; PROJECT NO: Z-1772; TASK NO: ET08; WORK UNIT ACCESSION NO:

11 TITLE (Include Security Classification)
BATMAN (Battle-Management Assessment System) & ROBIN (Raid Originator Bogie Ingress): Rationale, Software Design, and Database Descriptions

12 PERSONAL AUTHOR(S)
Federico, P-A., Bickel, S. H., Ullrich, R. R., Bridges, T. E., & Van de Wetering, B.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Technical Note	FROM 87 Apr TO 89 Apr	1989 April	

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Computer-based performance measurement, computer simulation, wargaming, and battle management
05	09		
15	12		

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This technical note contains the rationale, software design, and database descriptions for BATMAN (Battle-Management Assessment Systems) & ROBIN (Raid Originator Bogie Ingress). These software systems are being designed and developed to assess how well individuals can allocate, deploy, and manage air, surface, and/or subsurface tactical assets during simulated sea battles in many warfare areas. Together they form a desk-top, computer-based, performance-measurement system incorporating high resolution graphics and low level modelling to fill the gap between board games that are run in real or fictitious time with subjective assessment and inappropriate feedback and very expensive and manhour-intensive, mainframe-based simulators. Two of the major contributions of these dual systems are a very friendly human-computer interface and automated performance assessment.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Pat-Anthony Federico	22b TELEPHONE (Include Area Code) (619) 553-7777
	22c. OFFICE SYMBOL Code 15

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

FOREWORD

This advanced development was performed under project Z1772-ET008, Computer-Based Peformance Testing, sponsored by the Deputy Chief of Naval Operations (Manpower, Personnel, and Training). The general goal of this effort is to develop computer-based simulations of operationally oriented job-sample tasks in functional contexts, and determine if these state-of-the-art assessment schemes result in better performance measurement than traditional testing techniques.

The part-time assistance of a number of former and present university students is acknowledged in developing and evaluating several versions of BATMAN & ROBIN software. These include Fred Buoni, Chris Cassella, Bill Kamm, Bill Limm, Nina Liggett, Glen Little, Robert McCarter, Tony Meadors, Lorna Mildice, Dan Nadir, Alex Olender, Regina Peck, Jeff Roorda, Chris Ryan, Karl Schricker, Ellen Schuller, Alice Shimada, and Brian Smithey.

E. G. AIKEN

Summary

This technical note contains the rationale, software design, and database descriptions for BATMAN (Battle-Management Assessment System) & ROBIN (Raid Originator Bogie Ingress).

Part I: Rationale

Background

Customary methods for measuring performance either on the job or in the classroom involve instruments which are primarily paper-based or procedures which are judgmentally very subjective, e.g., check lists, rating scales, mastery demonstrations, criterion- and domain-referenced tests, critical incidents, disguised or unobstrusive observations. Many of these methods do not measure real-world, operationally oriented, job-performance tasks with sufficient fidelity, reliability, and/or validity. Consequently, evaluation at its best is somewhat suspect, and decisions based upon this kind of assessment may be erroneous. Better testing techniques are needed for assessing Navy trainees against performance criteria employing tasks functionally similar to those encountered in operational contexts. One attempt to fulfill this requirement involves the use of computer technology which is rapidly appearing in a number of Navy training and testing environments. Although these systems have been used on a small scale for various courses, widespread use necessitates advanced development of innovative, state-of-the-art, computer-based testing systems.

BATMAN & ROBIN

BATMAN & ROBIN are being designed and developed to assess how well individuals can allocate, deploy, and manage air, surface, and/or subsurface tactical assets during simulated sea battles in many warfare areas. Together they form a desk-top, computer-based, performance-measurement system incorporating high resolution graphics and low level modelling to fill the gap between board games that are run in real or fictitious time with subjective assessment and inappropriate feedback and very expensive and manhour-intensive, mainframe-based simulators. Two of the major contributions of these dual systems are a very friendly human-computer interface and automated performance assessment.

Because of the nature of their software (i.e., highly structured, very modular, practically object-oriented) and databases (i.e., independent of simulation programs, platforms easily added or deleted, parameters readily alterable), BATMAN & ROBIN could be used to complement (a) tactical training and testing; (b) assist tactical and systems development, analysis, and evaluation; (c) and aid tactical planning and decision making. That is, BATMAN & ROBIN lend themselves to the incorporation of higher fidelity computer models and classified databases. Consequently, these software systems could also be employed as user friendly frontends to sophisticated simulation models and complex databases.

Improved capabilities and performance enhancements are presently being added to BATMAN & ROBIN. The current software release is for demonstration, evaluation, and feedback purposes only, i.e., beta testing. At this time, these systems are not to be used for tactical training or testing as well as tactical planning or decision aiding. The databases are sanitized or unclassified, and platform parameters and computer models are only approximate.

Part II: Software Design

The second part of the documentation deals with BATMAN & ROBIN software design philosophy, program composition, Sun graphic utilities, object-definition-database and graphics editors, global data structures, and descriptions of software packages.

Part III: Database Descriptions

The third part of the documentation describes BATMAN & ROBIN's object-definition, graphic, scenario, and user-performance databases.

Contents

Part I: Rationale

Part II: Software Design

Part III: Database Descriptions

List of Tables

List of Figures

Part I: Rationale

1.0 Background

Customary methods for measuring performance either on the job or in the classroom involve instruments which are primarily paper-based or procedures which are judgmentally very subjective, e.g., check lists, rating scales, mastery demonstrations, criterion- and domain-referenced tests, critical incidents, disguised or unobtrusive observations, and multiple-choice, completion, true-false, and matching formats. A number of deficiencies exist with these traditional testing techniques: (a) biased items are generated by different individuals, (b) item-writing procedures are usually obscure, (c) there is a lack of objective standards for producing tests, (d) item content is not typically sampled in a systematic manner, (e) there is often a poor relationship between what is taught and test content, and (f) the infusion of halo, primacy, and recency effects, stereotypes, and implicit personality theories frequently distort assessment.

Many of these methods do not measure real-world, operationally oriented, job-performance tasks with sufficient fidelity and reliability. It is difficult for such instruments to estimate some job-related performances in functional contexts with adequate validity to warrant claims of correct assessment. Consequently, evaluation at its best is somewhat suspect, and decisions based upon this kind of assessment may be erroneous. This could result in either overtraining which increases costs needlessly, or undertraining which culminates in unqualified graduates being sent to the fleets. Better testing techniques are needed for assessing Navy trainees against performance criteria employing tasks functionally similar to those encountered in operational contexts.

One attempt to fulfill this requirement involves the use of computer technology which is rapidly appearing in a number of Navy training and testing environments. Although these systems have been used on a small scale for various courses, widespread use necessitates advanced development of innovative, state-of-the-art, computer-based testing systems. Technological and operational problems associated with the pervasive implementation of these systems to assess performance must be ascertained. However, there is no suitable knowledge base which can be tapped by the Navy, or others, for developing, evaluating, selecting, and using computer-based strategies incorporating graphic representations of job-sample tasks.

What is required is a theoretically and empirically grounded technology of producing procedures for performance assessment which will correct deficiencies associated with traditional testing techniques as well as provide a sufficient knowledge base for intelligently using computer-based measurement. State-of-the-art, computer-based systems are needed to support the preparation, administration, evaluation, and interpretation of operationally oriented tests. Such systems could provide capabilities for creating and evaluating generic testing strategies to assess student performance in functional contexts using computer-based graphic simulations, models, or metaphors as well as accessing generalizable and transferable software tools to implement prescriptive procedures to assist in the development of job-sample tests. The development, evaluation, and implementation of

computer-based-test-production systems are needed to determine the degree of support such technology can give to Navy instructional development personnel and training managers. The use of automated aids, software tools, and general procedures should result in noticeably increased effectiveness and efficiency in the performance-test-production process.

Measurement strategies grounded upon graphic simulations of job-relevant tasks should permit more accurate and valid evaluation of combat oriented performance since these should reflect requisite readiness standards and criteria more closely than traditional testing techniques. The capability to produce improved assessment strategies for use in Navy readiness training should assist commands in avoiding the twin problems of over training and under training. The development and implementation of computer-based-performance-assessment systems should sustain and improve the evaluation and estimation of the combat-effectiveness skills of individuals, teams, and crews. This is especially true in those situations in readiness and school-house training and testing where exercising job-relevant skills will be precluded because of the nature of the operational tasks themselves, i.e., they involve hazards, they are drawn-out over time, and/or they occur infrequently. These innovative techniques should culminate in better standardization in the output of Navy training pipelines as well as diagnostic capability for job-sample measurement of the combat effectiveness of individuals, teams, and crews.

Assessment strategies grounded upon computer-based graphic simulations of job-relevant tasks should enable more realistic evaluation of operationally oriented performance and transfer of training than is currently the case. It is expected that these state-of-the-art systems for assessing students' potential operational performance in functional contexts will substantially improve the reliability and validity of Navy performance testing. Very few data are available regarding the psychometric properties of testing strategies using computer-based graphically represented simulations, models, or metaphors (Federico, 1989; Federico & Liggett,1989; Liggett & Federico, 1986; Little, Maffly, Miller, Setter, & Federico, 1985). Technical information is needed concerning the accuracy, consistency, sensitivity, fidelity, and utility of these computer-based assessment schemes compared to more traditional testing techniques.

2.0 BATMAN & ROBIN

Within this framework, the Battle-Management Assessment System (BATMAN) and its counterpart, Raid Originator Bogie Ingress (ROBIN), are being created as part of the Computer-Based Performance Testing project. This advanced development work is attempting to determine empirically whether some computer-based, graphically represented job-sample tests significantly improve the state-of-the-art of operationally oriented performance assessment. BATMAN & ROBIN were designed and developed as a desk-top, computer-based, performance-measurement system incorporating high resolution graphics and low level modeling to fill the gap between board games that are run in real or fictitious time with subjective assessment and inappropriate feedback and very expensive and manhour-intensive, mainframe-based simulators.

Such a system could complement traditional training and testing techniques by (a) serving as

an original learning experience, (b) reinforcing instructional objectives that have already been acquired from other media, (c) evaluating the understanding and use of warfare theories, principles, rules, and procedures, (d) monitoring, diagnosing, and feeding back gaps in the decision-making performance, (e) providing insight into the complex cognitive processes involved in managing battles, and (f) teaching trainees to play different adversarial roles with multiple conflicting objectives.

ROBIN is a very user friendly, animated, computer-based, graphic simulation, model, metaphor, or microworld. It was initially designed to allow the creation of a large number of Red force raids involving different types, numbers, formations, flight paths, and tactics of missile-launching threat bombers that are attacking a Blue carrier-based task force which can be located in the many oceans of the world. Subsequently, ROBIN was enhanced to include different classes of hostile surface and subsurface platforms which can be configured as several surface action groups (SAGs) and/or wolf packs or plugs. ROBIN permits an individual who is specifying a scenario or test item to designate the nature and number of Blue force tactical resources. These will be available to the person who will be subsequently performing in BATMAN by allocating, deploying, and managing air, surface, and/or subsurface assets against multiple threats. In addition to a generator, ROBIN includes a scenario viewer and editor. The viewer allows an individual who has created a scenario to see it as a testee, student, or other user would see it in BATMAN. The editor permits the random access of a scenario which has been generated and stored previously for any warfare area and the modification of it without having to reproduce all of its facets. ROBIN gives the individual who is generating the scenario the ability to place different planes and weapons on large or small aircraft carriers, land bases, or battle ships so that these will be presented later on to the person who is wargaming in BATMAN. ROBIN could also be used as a scenario generator independently of BATMAN. That is, it could be adapted to frontend Navy systems such as TACDEW, BFIT, NAVTAG, and REAS.

BATMAN is also a very user friendly, computer-based, animated, graphic simulation or microworld. It was initially designed to assess how well individuals allocate, deploy, and manage tactical aircraft for the outer air battle to defend carrier-based task forces against incoming, missile-launching, Soviet bombers in various warfare theaters yielding many different scenarios. In BATMAN the task force, consisting of a large or small aircraft carrier or battle ship and their escort ships and submarines, has to be deployed in a warfare theater against hostile platforms. Cyclic or flex deck flight operations have to be planned and executed. Fighters and attack aircraft have to be (a) loaded out with weapons, (b) assigned to combat air patrols (CAPs) and chainsaws, (c) placed on ready 5, 15, or 30 alerts, (d) rendezvoused for in-flight refueling, (e) deck launched, and (f) returned to home base. Tankers and early warning aircraft can be (a) placed on alert states, (b) prepositioned in the defensive grid, (c) launched and recovered during the battle, and (d) controlled prior to, or during, the simulated engagement. Fighters can detect and intercept threat aircraft and missiles; attack aircraft can detect and launch weapons against hostile surface combatants. Platform status or fixment, radar or sonar coverage, as well as other information, can be displayed for the individual managing the battle in BATMAN. Multiple carrier-based, battle-ship-based, or land-based task forces can be created in ROBIN and employed in BATMAN. BATMAN measures performance automatically and objectively against sixteen criteria which are immediately fedback to the individual at the end of each scenario, e.g., the average

distance from the defended point threat aircraft were detected, the percentage of hostile bombers destroyed, the percentage of aircraft lost because of lack of in-air refueling. These multivariate performance measures are saved in files by the system for subsequent statistical analyses, and are available for both formative and summative evaluations of performance (Bloom, Hastings, & Madaus, 1971). This automated assessment ability has been expanded to permit performance measurement in multithreat scenarios, i.e., simultaneous surface, subsurface, and/or air warfare, involving as many as three Blue carrier-based, battle-ship-based, and/or land-based task forces and several Red SAGs and wolf packs as well as a very large number of long range bombers and tactical fighters.

The unclassified or sanitized database that BATMAN & ROBIN use to represent different parameters or characteristics of Blue and Red air, surface, and subsurface platforms together with their sensor and weapon systems is independent of the software used to execute the simulation or wargame. This property permits individuals to add, delete, and change easily and arbitrarily platforms and their corresponding systems employed in the wargame. Also, this feature facilitates the adoption of existing classified databases. Currently, BATMAN & ROBIN approximate attributes such as the quantities of fuel different Blue aircraft normally carry, the amount they use at catapult launch, the rates of fuel consumption at "max conserve" and full military power. Also, they model at a low level of fidelity radar and sonar coverage for air, surface, and subsurface Blue platforms, and launch acceptability regions for missiles together with their associated probability of kills. Red air platforms in BATMAN & ROBIN have the capabilities to raid in stream or abreast, vary altitude and speed, place chaff corridors, jam communications, and launch antiship missiles.

With the insertion of a classified database and higher fidelity simulation models, BATMAN could be used to complement the tactical training and testing of individuals who are in the pipeline for, or who already are, air warfare officers, air resources officers, tactical action officers, tactical air and anti-air warfare commanders and coordinators, staff tactical watch officers, crews of early warning, fighter, and attack aircraft, as well as others who must be familiar with the conduct of the outer air battle or surface and subsurface warfare on ships, planes, and submarines. By incorporating complex computer models and a classified database, BATMAN & ROBIN could also be used to (a) develop and evaluate tactics themselves, (b) aid tactical planning and decision-making, and (c) assist in the analysis and appraisal of sensor, weapon, and communications systems. The human-computer interface, the automated performance-measurement, the independent database, and the software design contribute to the potential extrapolation and adaptation of BATMAN & ROBIN to uses which may be obvious to experienced naval officers as well as warfare and system analysts, but not the originator.

BATMAN & ROBIN are still under development. Currently, more sophisticated underlying operational models are being conceptualized and incorporated in BATMAN. These deal with anti-air, antisubmarine, and electronic warfare as well as a communications network, the Joint Tactical Information Distribution System. Approximately fifty more aircraft as well as surface and subsurface combatants for Blue and Red forces, and twenty weapons of different types, have been added to these systems bringing the total number of platforms and weapons to about seventy and thirty, respectively. If the user desires, more platforms and their corresponding weapons and sensors can be easily added, or some deleted, without

rewriting any of the simulation program because of the modular software and independent database. Many other improved features and performance enhancements have been incorporated into BATMAN & ROBIN to better meet Navy and Marine Corps needs. The present software release is at testbed and transition sites for demonstration, evaluation, and feedback purposes only. AT THIS TIME, BATMAN & ROBIN ARE NOT TO BE USED FOR TACTICAL TRAINING OR DECISION AIDING. THE DATABASE EMPLOYED IS SANITIZED OR UNCLASSIFIED; PLATFORM PARAMETERS AND COMPUTER MODELS ARE ONLY APPROXIMATE.

Both BATMAN & ROBIN are written in the "C" programming language (Kernighan & Ritchie, 1978) and run on SUN-4/260C families of computers under SUN's 4.0 release of the UNIX operating system (McGilton & Morgan, 1983; SunOS Reference Manual, 1988). The software was designed and developed so that it is structured or layered using independent modules and object-oriented programming concepts (Stroutstrup, 1988; Wilson, 1988) when suitable. This was done wherever appropriate to instill the following salient features in the software: generalizability, transferability, understandability, modifiability, extensibility, and reusability. The use of generic structures, object-oriented techniques, program modularity, mnemonic variables, and top-down design made it easier to accommodate these goals. Flexibility (ease of changing, expanding, and upgrading) of the software was of paramount importance in this endeavor. These important software features of BATMAN & ROBIN make it feasible to (a) adapt them readily to generate scenarios involving many distinct surface, subsurface, air, and land platforms; (b) use them in the training and testing of battle managers in a number of different warfare areas, e.g., antisubmarine, terrestrial environments; and (c) adapt them for aiding in the analysis and evaluation of weapon, sensor, and communication systems assuming that sophisticated and validated computer models and relevant and widely acceptable databases are used. BATMAN & ROBIN employ a direct-manipulation, human-computer interface (Hutchins, Hollan, & Norman, 1986; Shneiderman, 1983) where graphic objects, e.g., aircraft or ship silhouettes, are continuously depicted, moved, and queried by the operator physically moving and clicking a mouse resulting in immediately visible impact on the icons.

This documentation describes the software design and databases of BATMAN & ROBIN, version 3.0. It is intended for software engineers familiar with Unix (McGilton & Morgan, 1983; SunOS Reference Manual, 1988), SunView (SunView Programmer's Guide, 1988; SunView System Programmer's Guide, 1988), and the C programming language (Kernighan & Ritchie, 1978). Part II of this document covers BATMAN & ROBIN's software design, including descriptions of the software's data structures and packages. Part III covers BATMAN & ROBIN's databases, including the Object-Definition, Scenario, Graphics, and User databases. Follow on documentation will discuss the human-computer interface (Federico, Bickel, Ullrich, & Bridges, in preparation) and simulation models employed in BATMAN & ROBIN.

2.1 Notational Conventions

The following notational conventions are used throughout this document:

Convention	Meaning
Bold	UNIX filenames, C package names, Object- Definition Database parameters, and document section titles are set in bold type to distinguish them from ordinary text. In this documentation, "package" is used to refer to a collection of related C functions and data types grouped in one or more files.
Italics	Italics are used for the names of C functions and variables. In addition, italics are occasionally used to emphasize particular words in the document.
ITALIC CAPITALS	Italic capital letters are used for the names of C data structures.

Part II: Software Design

3.0 Purpose and Scope

This part of the documentation provides an overview of BATMAN & ROBIN software. It includes descriptions and diagrams of the system's software packages, data flow, and data structures, and can be used as an aid by those interested in modifying and enhancing the system. The next section provides an introduction to the adopted software-design philosophy, together with some guidelines to follow when attempting to modify the software. The remaining sections in this part of the documentation describe each software component of BATMAN & ROBIN, concluding with global data structures and specific packages.

Each software package description contains a list of the functions exported by the package, thereby providing an overview of the package's interface. For documentation on specific functions, refer to the commented code in the BATMAN & ROBIN software library in the directory: **/nprdc/wargame/batman**. Also, algorithms and interface specifications for each C function can be found in the directory: **/nprdc/wargame/batman/docs**. The files in this directory are named **xxx.d**, where **xxx** is the name of the corresponding C package.

4.0 Introduction

To help provide a better understanding of BATMAN & ROBIN software, this section outlines the design philosophy and system guidelines.

4.1 Design Philosophy

The intent of the human-computer interface is to provide intuitive ease and flexibility in constructing and gaming tactical scenarios. To achieve these goals and allow for future

system expansion, much effort has gone into creating generic data structures, modularized software, object-oriented-programming style, and functional standards that provide a natural mapping between the system's human interface and the code-level design. These principles of the interface and the software packages should be maintained by all persons interested in contributing to this system.

4.2 Software Caveats

Before attempting to modify the software, be aware of the following:

- BATMAN & ROBIN are currently under development. Modifications and enhancements are made continually. Changes made by others will not be supported by the original developers. These include alterations to (a) the direct-manipulation human-computer interface, (b) the format and contents of scenario data files generated by ROBIN and executed by BATMAN, (c) the algorithms, code, and databases involved in platform behavior, (d) the database editing tools, (e) the windowing system's structural design, (f) the code and databases used for displaying maps, and (g) the grid coordinate system.

- BATMAN & ROBIN form a complex system. Tuning the software to run efficiently with different databases and algorithms requires an understanding of several components. Software modifications may appear feasible, but without knowing the limitations of the graphics and computational structures, they may become burdensome or impossible tasks. Many of these issues are beyond the scope of this document and direct consultation with the developers is advisable.

- Contact NPRDC before changing BATMAN & ROBIN. Many modifications may have already been scheduled for a future date, rejected because of limited customer interest, or suspended because of technical problems in the system's design.

4.3 Guidelines for Adding a Computer Model

BATMAN & ROBIN software has been designed and developed as a generalizable, object-oriented, modular system. This readily lends itself to the addition of other computer models to increase the level of fidelity or functionality of BATMAN & ROBIN. To successfully incorporate a new simulation model, some programming guidelines must be followed, specifically:

- The added model must be written in a language compatible with the C programming language and the Sun-4 series computers, such as FORTRAN 77 or Pascal.

- The new model must consist of functions with well defined software interfaces that are independent of, and compatible with, BATMAN's existing simulation models.

- Any input data the added model requires must be completely separate from the

existing BATMAN & ROBIN databases.

- Any output produced by the new model must be initialized and generated independently from BATMAN's output.
- The computer model must not require any user interaction, and it must operate only on data from files or parameters passed to it from BATMAN.
- For the mouse to track smoothly, it must be serviced once every 15 milliseconds. To accommodate this requirement, the model's calculations must be sectioned into reasonable pieces so it can return to BATMAN within this time interval.
- If the simulation model requires functional changes to the existing BATMAN software, these changes must be coordinated with, and performed by, NPRDC.

It is recommended that the added computer model's interfaces or hooks in BATMAN & ROBIN software be written by NPRDC. Since NPRDC is intimately familiar with the software, these interfaces can be implemented efficiently and effectively. Moreover, it eliminates the need for the modeler to have extensive knowledge of the internals of BATMAN & ROBIN.

Because the complexity of simulation models vary dramatically, NPRDC cannot be responsible for the performance of BATMAN & ROBIN after these models have been incorporated. The current version of the software has been highly optimized for speed. Adding a compute-intensive model will likely degrade BATMAN's performance.

5.0 Software Composition

BATMAN & ROBIN are composed of four databases: Object-Definition, Graphics, Scenarios, and Users; and three processes: BATMAN & ROBIN, Object-Definition, and Graphics Editors. This software composition is illustrated in Figure 1.

Although BATMAN & ROBIN are truly one process, they are often conceptually considered as two separate processes, as in Figure 1, since they serve specific purposes.

The Object-Definition Editor is used to create and modify the attributes of objects in the Object-Definition Database, e.g., the fuel-consumption rates of air platforms. The Graphics Editor is used to create and modify BATMAN & ROBIN's icons. ROBIN is used to create and modify simulated scenarios that BATMAN will later present to its users. The Sun Graphic Utilities are used to provide a friendly interface to BATMAN & ROBIN. For a more detailed description of the databases, refer to Part III: Database Descriptions.

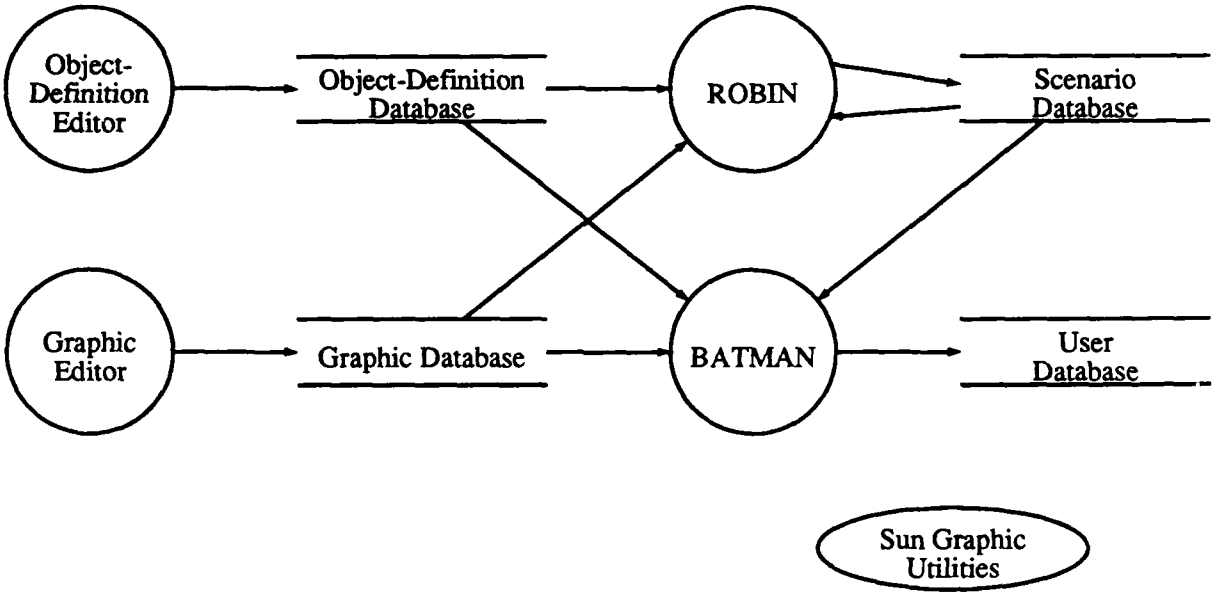


Figure 1. BATMAN & ROBIN Software Composition

6.0 Sun Graphic Utilities

For comprehensive understanding of BATMAN & ROBIN software, the reader must be familiar with the SunView programming environment (SunView Programmer's Guide, 1988; SunView System Programmer's Guide, 1988; Pixrect Reference Manual, 1988). The tools provided by SunView are incorporated throughout BATMAN & ROBIN, and often dictate the design of the system. This section will provide a brief overview of some of the SunView features used by BATMAN & ROBIN.

BATMAN & ROBIN incorporate three SunView window types: Frame, Canvas, and Panel. The Frame is a parental window serving as a border for the Canvasses and Panels. The Canvasses are the background windows where the maps, grid, and platforms are drawn. All other windows in BATMAN & ROBIN are Panels. The Panel is an operator-interface mechanism that allows the programmer to create icon and pull-down menus, button selections, and slider, text, and cycle items. Figure 2 provides an example of these window types. Note that BATMAN & ROBIN do not use any SunView Sub-Frames because they require an extra UNIX device.

When an application creates a SunView Panel, it must define the characteristics and location of the Panel. Items can then be assigned to the Panel. Just like the Panel itself, each item within the Panel has its own characteristics and location. Each Panel item which can receive user input (e.g., a choice item labeled with an icon of an F-14 fighter) is assigned a notify function that is executed when the Panel item receives the appropriate user input from the mouse.

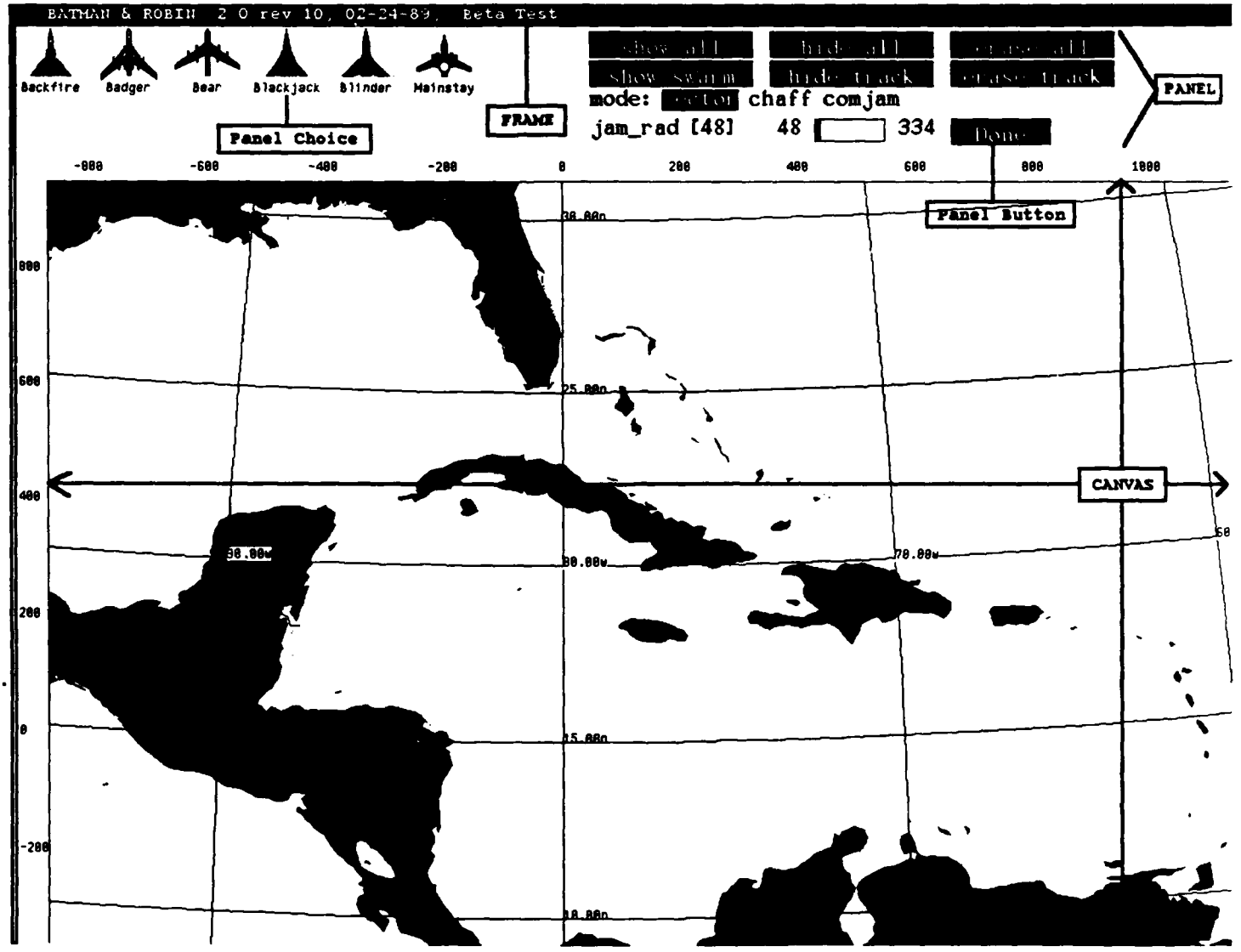


Figure 2. Sun Window and Panel Types

The SunView notifier (SunView Programmer's Guide, 1988) is a package of interrupt-driven functions that manage window input and output, and execute user-specified processes. User input, window drawing, and other window events are processed and routed to the appropriate window by the notifier. If a Panel receives an input event, it will route it to the appropriate Panel-item's notify routine. Figure 3 illustrates this process.

This is an important concept since it greatly influences the structure of SunView applications. Many of BATMAN & ROBIN's features are partitioned into Panel create functions and Panel notify functions.

suntool, **sunwindow**, and **pixrect** libraries contain the SunView graphics and windowing functions. They must be linked with BATMAN & ROBIN's object code.

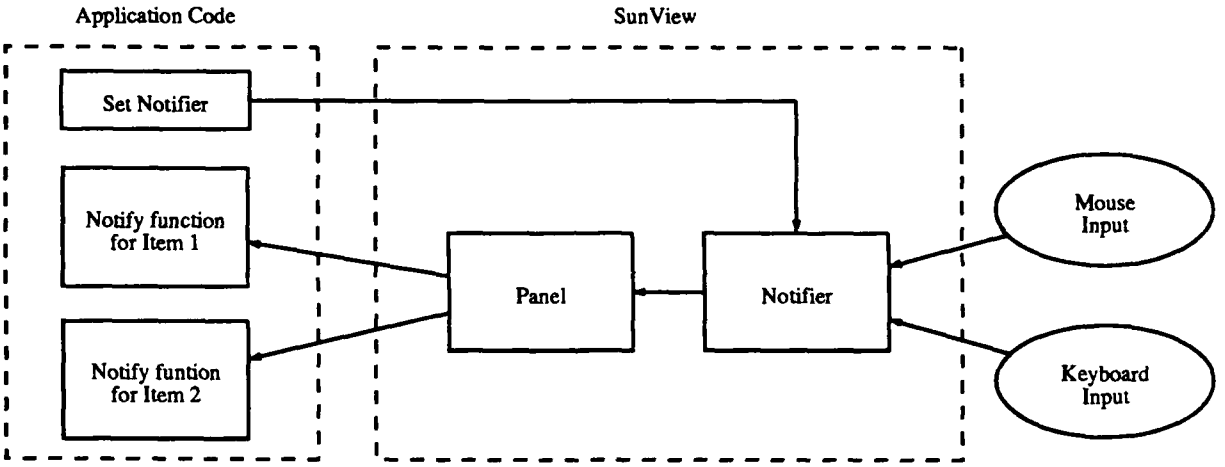


Figure 3. SunView Notifier

7.0 Object-Definition Database Editor

The Object-Definition database is implemented as an ASCII text file. Any word processor that creates strict ASCII files can be used as the Object-Definition Database Editor. The developers have used the vi editor, a standard Unix utility (Sun UNIX Commands Reference Manual, 1988). The contents of the Object-Definition database are described in Part III of this document.

8.0 Graphics Editor

The Graphics editor can be any method for creating icons or graphic objects. The graphics

supported by BATMAN & ROBIN are Sun standard rasterfiles (Pixrect Reference Manual, 1988). All icons and graphic objects should be black-and-white as well as one-bit deep since BATMAN & ROBIN apply their own colors to these objects. Commercial Sun rasterfile editors have been used, and a custom software program has been developed by NPRDC to create icons. None of these programs are discussed in this documentation.

9.0 BATMAN & ROBIN Global Data Structures

Table 1 lists global data structures that are used throughout the BATMAN & ROBIN software, and identifies the header file where each data structure is defined. Most of the structures are used as scratch pads when scenarios are built with ROBIN, and all of the structures are used for storage when scenarios are presented by BATMAN.

Table 1.

Data Structure to File Mapping

Data Structure	File	Contents
SCENARIO_HEAD	**scenario.h**	scenario-related data
THEATER	**scenario.h**	scenario log
PATH_FORCE	**force.h**	red-force resources
CONSOLE_FORCE	**force.h**	blue-force resources
STATUS_REC	**force.h**	sibling or sub resources
TYPE_REC	**force.h**	platform type definition
PLATFORM	**platform.h**	one platform
PROJECTILE	**platform.h**	one projectile
CHAIN_REC	**platform.h**	one chainsaw
CHAIN_NODE	**platform.h**	one chainsaw node
DETECT_NODE	**platform.h**	one detection-unit
DETECT_PROJ	**platform.h**	detection-unit's weapons

Memory pointers are used to tie these structures together, taking away the need for duplicate copies of structures. For example, all *PLATFORM*s in the game, have a pointer back to the force that they belong to, either *PATH_FORCE* or *CONSOLE_FORCE*.

Figure 4 illustrates the organization of the these data structures during a typical BATMAN scenario. The structures are built after the scenario to be presented becomes known. The *SCENARIO_HEAD* structure is at the top and contains pointers to the *THEATER*, *PATH_FORCE* and *CONSOLE_FORCE* structures. The *SCENARIO_HEAD* structure has also become a depository for all miscellaneous global variables in BATMAN & ROBIN.

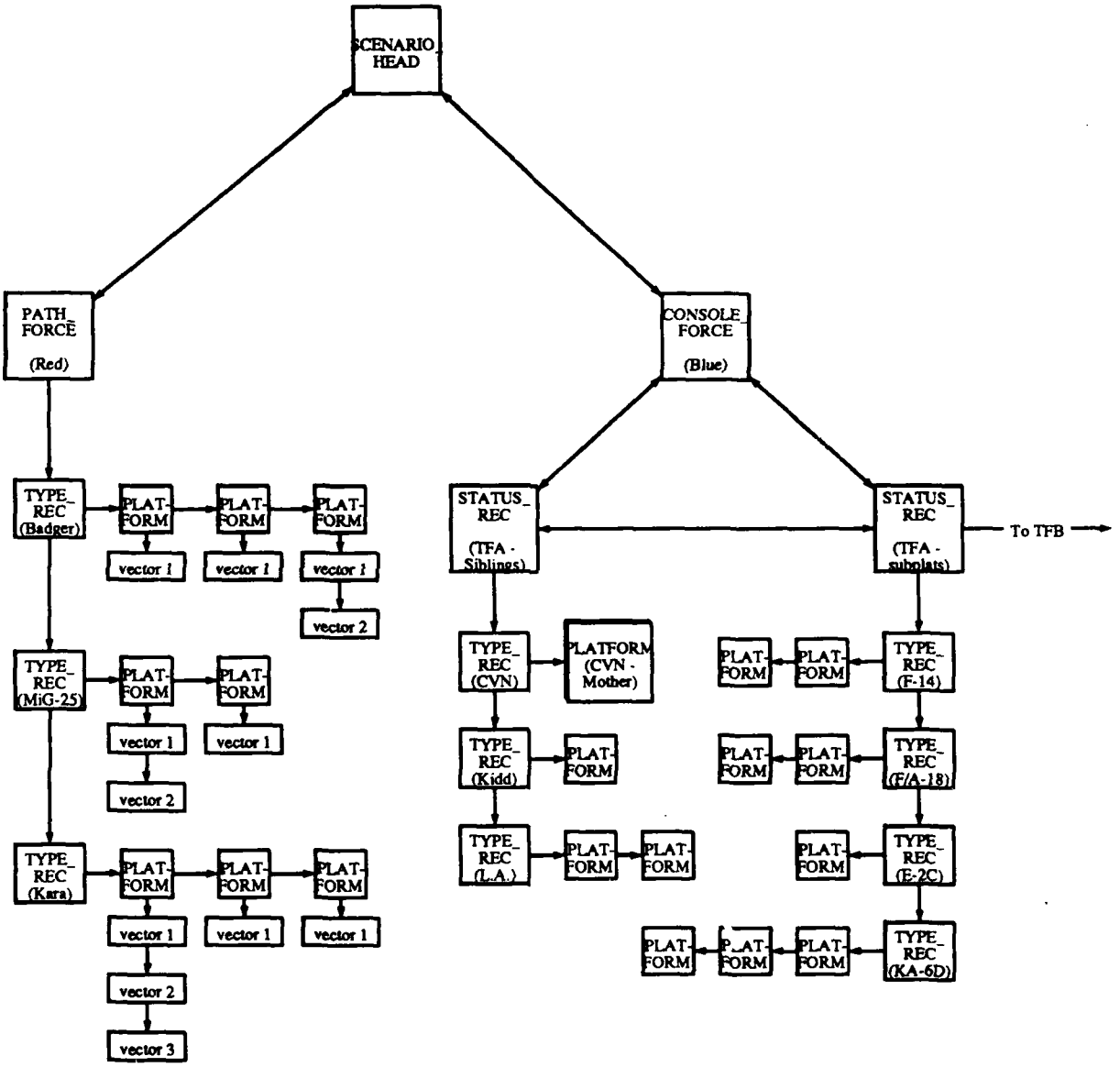


Figure 4. BATMAN & ROBIN Data Structure Organization

The *THEATER* record contains a log of all scenarios currently defined, and is discussed in greater detail in the **Scen_db_access** subsection below.

The *PATH_FORCE* contains the Red-force attack, and has a list of *TYPE_REC*s for each different type of platform in the raid, e.g., a Badger aircraft. Attached to each *TYPE_REC* is a list of *PLATFORM*s of that type. There is one *PLATFORM* structure for each platform in the raid. For example, if the attack force had four Badger aircraft, there would be one *TYPE_REC* for the Badger and four *PLATFORM*s, one for each Badger. Attached to each Red-force *PLATFORM* is path information describing how the platform moves in BATMAN.

The *CONSOLE_FORCE* contains all Blue-force tactical resources that are organized into one to three task forces: Task Force Alpha (TFA), Bravo (TFB), and Charlie (TFC). Each task force, which can be centered on a large or small aircraft carrier, battle ship, or land base, is then divided into *sibling* and *sub* platforms. Information specific to the *sibling* and *sub* categorizations is contained in *STATUS_REC*s. Each *STATUS_REC* contains a list of *TYPE_REC*s for each different type of platform in the group. Then, in a similar manner to the *PATH_FORCE* structure, a list of *PLATFORM*s is attached to each of these *TYPE_REC*s.

One of the platforms in the *sibling* category must be a *mother* or *home-base* platform, e.g., an aircraft carrier. Note that the three terms *mother*, *sub*, and *sibling* refer to generic platform types. A *mother* platform is the primary one in a task force, carrying *sub* platforms, and accompanied by *sibling* platforms. A *mother* platform may be a large or small aircraft carrier, a land base, or something else, e.g., Iowa class battle ship. A *sub* platform might be an aircraft, e.g., F-14 or a weapon, e.g., Phoenix missile. A *sibling* platform might be a Ticonderoga class cruiser or a Los Angeles class submarine.

Figure 5 illustrates a *PLATFORM* data structure containing data for an F-14.

This F-14 *PLATFORM* structure contains:

- pointers back to the appropriate *TYPE_REC*, *STATUS_REC*, and *CONSOLE_FORCE* header,
- lists of the platform's weapons (*PROJECTILE*s) and detection-units (*DETECT_NODE*s and *DETECT_PROJ*s),
- flight information for the platform (*CHAIN_REC*s and *CHAIN_NODE*s),
- alert-level information for the air platform (*ALERT_HEAD* and *ALERT_LEVEL*s) - see section **Alert** below.
- Object-Definition and state variables, and
- a pointer to the next F-14 *PLATFORM* in list.

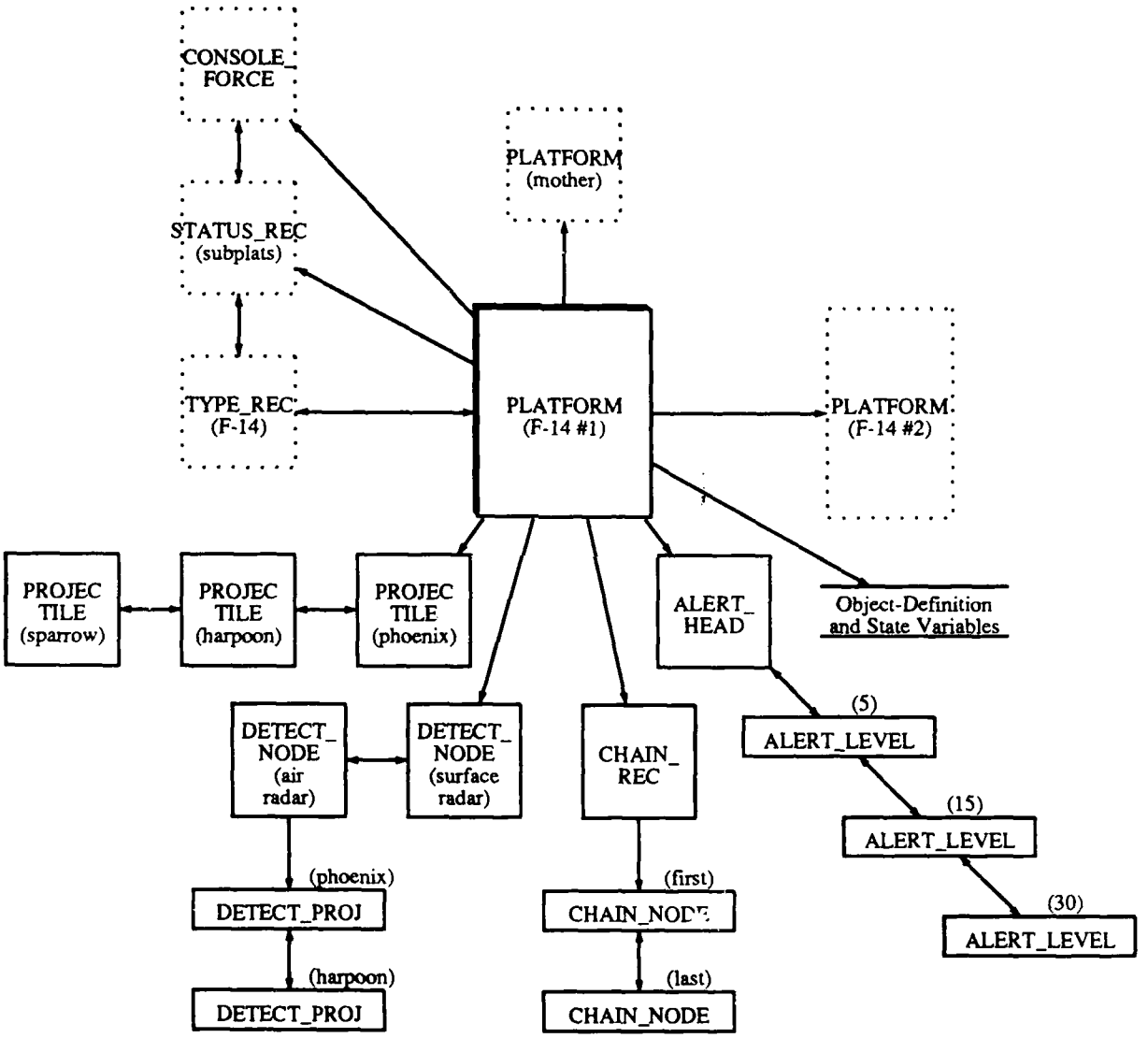


Figure 5. ***PLATFORM*** Data Structure

10.0 BATMAN & ROBIN Software

The following sections provide an overview of the BATMAN & ROBIN software. For purposes of this description, the software packages are organized into the following groups: (1) Initialization and Control, (2) Loadout, (3) Vector-Logic Grid, (4) Deployment, (5) BATMAN Simulation, (6) Performance Measures, (7) ROBIN, (8) World Database II, and (9) Tools. Figure 6 illustrates this grouping.

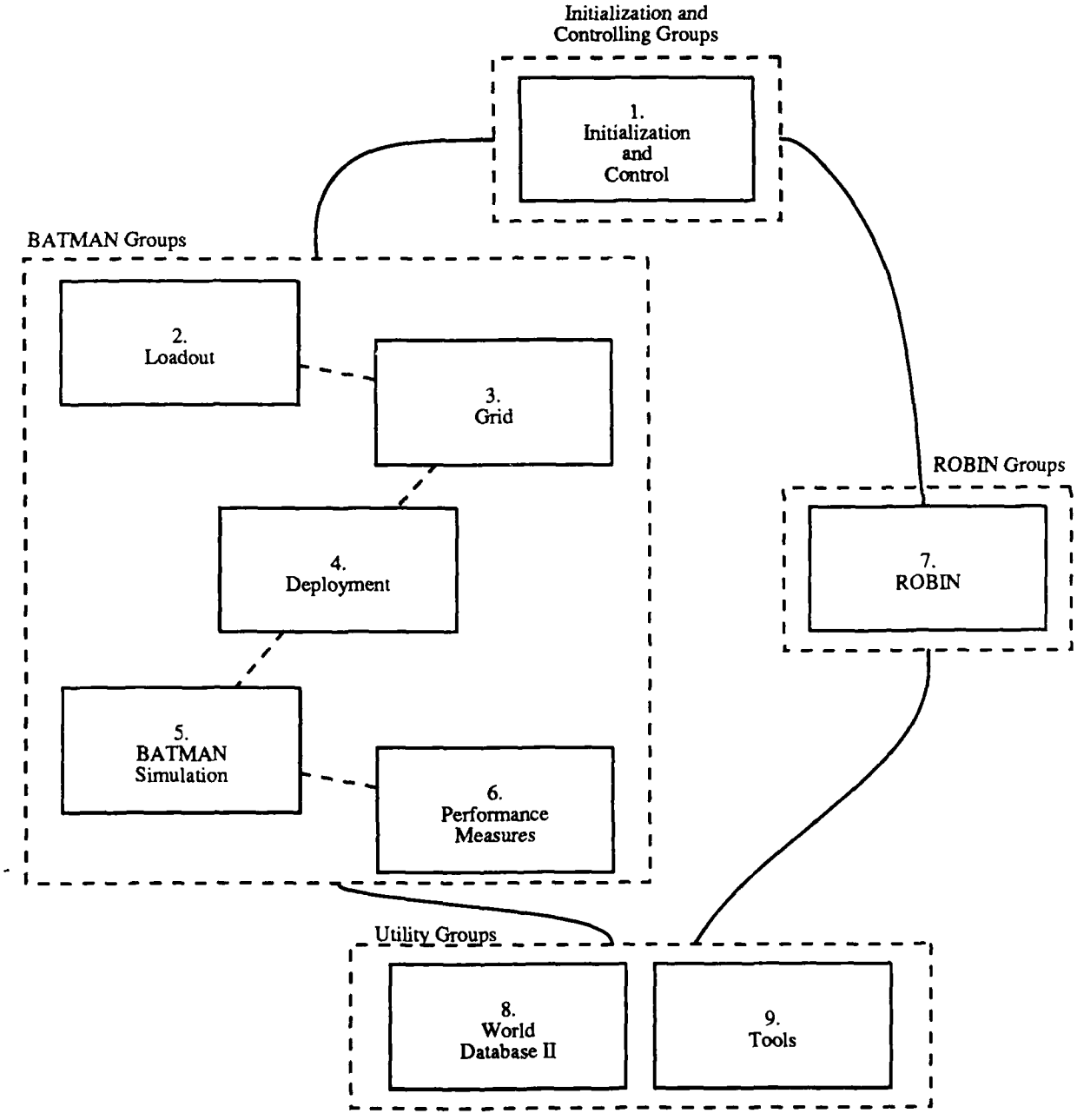


Figure 6. BATMAN & ROBIN Software Packaging

10.1 Initialization and Control Packages

The packages in this group setup the window environment, control the flow of BATMAN & ROBIN, initialize scenario data structures, and provide an interface to the user database.

10.1.1 Batman

This package contains the *main* function, and is the primary initialization and control package for BATMAN & ROBIN. This package also contains the declaration of the global *SCENARIO_HEAD* structure. Many of the functions that exist in this package are global utilities used by other BATMAN & ROBIN software packages.

Exported Functions:

main, *start_next_game_phase*, *set_stored_func*, *reset_stored_func*, *maincanvas_input_toggle*, *display_changes*, *define_maincanvas_rect*, *set_maincanvas_input_handler*, and *get_maincanvas_input_handler*.

10.1.2 Frontend

This package provides routines that rapidly display digitized photos of Blue and Red tactical platforms as a graphic introduction to BATMAN & ROBIN. The introduction is displayed if either "-f" or "F" appears on the command line.

Exported Functions:

frontend_graphics.

10.1.3 Init_con

This package initializes BATMAN & ROBIN's Console force data structures and panels. (At this stage of development, the Console force has been the Blue force. It is planned to increase the flexibility of BATMAN & ROBIN so that Red force can be a Console force too. This other option will take some time to develop. It is intended to complete this other feature in the near future.) The scenario number must be known before this package can be called, and is used to retrieve the appropriate scenario from the Scenario Database. This package reads in the scenario which specifies the types and numbers of platforms. Then, allocates records and fills them in with information obtained from the Object-Definition Database. This package will produce one *CONSOLE_FORCE* data structure, including all *STATUS_RECs*, *TYPE_RECs*, *PLATFORMs*, and *PROJECTILEs* that belong to the task force (refer to **BATMAN & ROBIN Global Data Structures**).

The high-level algorithm for building the *CONSOLE_FORCE* data structure is as follows:

```
Begin
    Get appropriate scenario from scenario database
    For (each task-force in the scenario)
        Allocate space for a CONSOLE_FORCE data structure
        Initialize appropriate fields of the CONSOLE_FORCE
```

```
        For (the sibling platforms that belong to the force)
            Allocate space for a STATUS_REC data structure
            Initialize appropriate fields of the STATUS_REC
            For (each different type of sibling platform)
                Allocate space for a TYPE_REC
                Initialize appropriate fields of the TYPE_REC
                For (each sibling platform of that type)
                    Allocate space for a PLATFORM data structure
                    Load PLATFORM with values from the Object-Definition
                        Database
                Endfor
            Endfor
        Endfor
        For (the sub platforms that belong to the force)
            **** Same logic as sibling platforms ****
        Endfor
    Endfor
End
```

Exported Functions:

init_console_force, *build_force_header*, *build_alert*, *build_status_rec*, *build_type_rec*, *build_sibling_plats*, *build_sub_plats*, *build_platforms*, *build_loadout_projectiles*, and *create_con_force_batman_panels_and_windows*.

10.1.4 Init_path

This package initializes the Path force data structures. (Currently, the Path force is the Red force or raiding force.) The scenario number must be known before this package can be called, and is used to retrieve the appropriate scenario from the Scenario Database. Also, it will build a list of flight-path information for each platform. Therefore, this package will produce one *PATH_FORCE* data structure, including all *TYPE_REC*s and *PLATFORM*s that belong to the raiding force (refer to **BATMAN & ROBIN Global Data Structures**).

This package is similar in function to **init_con**, but is slightly less complex since the *PATH_FORCE* data structure is simpler than the *CONSOLE_FORCE* data structure. The key differences between the two structures are that the *CONSOLE_FORCE* has *STATUS_REC*s distinguishing *sub* platforms from *sibling* platforms, the *PATH_FORCE* does not. Also, the *PATH_FORCE* has flight-path information for the predetermined Red attack, the *CONSOLE_FORCE* does not.

Because of these similarities with **init_con**, many of this package's **Exported Functions** have not been included in the following listing.

Exported Functions:

init_path_force.

10.1.5 Readobj

This package contains routines that will read platform, weapon, and detection-unit data from the Object-Definition Database. Both the **init_con** and **init_path** packages use these routines to build the *CONSOLE_FORCE* and *PATH_FORCE* data structures.

Exported Functions:

rd_plat_def, *rd_proj_def*, *build_projectiles*, *rd_detection_def*, and *build_detection*.

10.1.6 Scen_db_access

This package provides routines to build the *THEATER* data structure, and modify the **scenario_index** file of the Scenario Database. These routines are intended primarily for ROBIN. For more information on the Scenario Database, refer to Part III of this document.

One *THEATER* record for each warfare-theater is subsumed under the the *SCENARIO_HEAD*, and contains a log of all scenarios in that area. The data-dictionary definition for a *THEATER* is as follows:

THEATER =
- warfare-theater +
- num-scenarios-at-this-theater +
- list-of-the-scenarios-at-this-theater +
- title-for-each-scenario +
- list-of-red-platform-resources-for-each-scenario +
- list-of-blue-platform-resources-for-each-scenario

To date, warfare-theaters are, but not limited to, the following areas:

warfare-theater =
- Caribbean |
- Japan-sea |
- Bering-sea |
- Kamchatka-peninsula |
- South-east-asia |
- Arabian sea |
- Persian-gulf |
- Mediterranean |
- North-atlantic |
- Murmansk

Exported Functions:

read_scenario_data, *change_scenario_index_file*, and *get_next_scenario*.

10.1.7 User_db_access

This package provides routines to interface with the Users Database. For more information on the Users Database, refer to Part III of this document.

Exported Functions:

add_user, *del_user*, *user_exists*, and *rewrite_user*.

10.1.8 User_funcs

This package controls the human-computer interface when the user is performing one of the following functions:

- Listing all users in the User Database.
- Adding a user to the User Database.
- Deleting a user from the User Database.
- Selecting a user to run BATMAN & ROBIN.
- Selecting to run BATMAN & ROBIN in "demo" mode.

Hence, this package contains Panel create and Panel notify functions to coordinate the above activities. Access to these features is gained through the routine *user_funcs*, exported by this package. *user_funcs* will display a log of all users, and a Panel with the buttons: "Add User", "Delete User", "Demo", and "Exit". This routine remains active until principals, e.g., instructors, either select other individuals, e.g., students, as authorized users of BATMAN & ROBIN, choose to run BATMAN & ROBIN in "Demo" mode, or Exit from them.

Exported Functions:

user_funcs.

10.2 Loadout Packages

This group of packages creates the tactical-situation screen and handles weapons loadout for Blue-force air platforms.

10.2.1 Loadout_map

This package creates the tactical situation-screen, illustrated in Figure 7. At this stage of development, the tactical-situation display only contains the following:

- An orthogonal-projection map of the warfare-theater.
- Arrows representing air-threat axes.
- The location of Red-force surface and subsurface platforms.
- An icon for each Blue task-force.
- A message box with the defense condition or DEFCON.
- A Grid icon that, when selected, will leave loadout and advance to vector-logic grid definition.

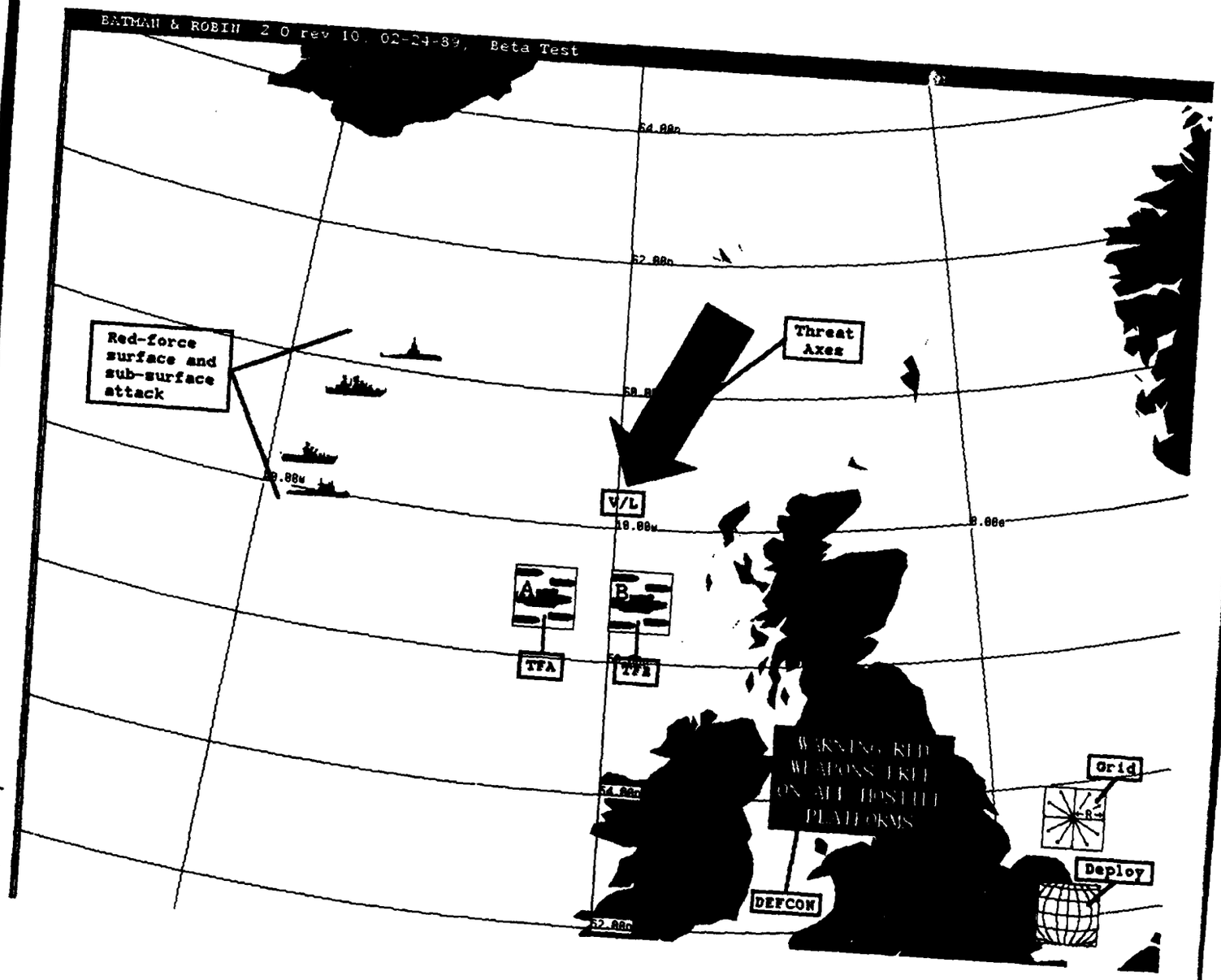


Figure 7. Tactical Situation Screen

- A Deployment icon that, when selected, will bypass grid definition and advance directly to deployment of tactical assets.

(Further development of the tactical situation will include other relevant DEFCONs besides RED, rules of engagement (ROEs), electronic warfare information, additional intelligence, sea state, and weather data. Incorporating DEFCONs WHITE and YELLOW into BATMAN & ROBIN, as well as ROEs, implies that Blue and Red platforms must be "smart", i.e., artificial intelligence techniques must be used to model these tactical situations properly.)

The scenario contains all of the information necessary to construct this screen.

Exported Functions:

show_scenario_location_map, *create_map_canvas_win_items*, *show_threat*, *show_threat_axises_and_craft*, *show_warning_type*, and *display_tf_panel_win*

10.2.2 Loadout_tf

This package contains functions for creating and interacting with Blue task-force display panels which contain large icons for the force's *home-base* or *mother* and *sibling* platforms. Refer to **BATMAN & ROBIN Global Data Structures** for a description of *mother*, *sibling*, and *sub* platforms. Task-force display panels are created by traversing the list of *TYPE_REC*s attached to the force's *sibling* platform *STATUS_REC*.

If the *mother* platform on a task-force display panel is selected, the user is "going aboard" the mother with the intention of loading weapons onto the task-force's air platforms. When this happens, control is passed to the **loadout** package discussed below.

Exported Functions:

create_tf_loadout_panel, *return_to_map_panel*, *show_tf_panel*, and *create_cf_panel_items*.

10.2.3 Loadout

This package contains functions for creating and interacting with Blue loadout panels which provide the human-computer interface for placing weapons on Blue air platforms. These panels are created by traversing the list of *TYPE_REC*s attached to the force's *sub* platform *STATUS_REC*.

Figure 8 illustrates a typical Blue loadout panel which is implemented with a *PANEL_WIN* (refer to **Panel_win** subsection below).

A Blue loadout panel contains the following six Panel Items:

1. An icon that returns to the Blue task-force display panel.
2. A paging icon used to display the next two *sub* platforms.

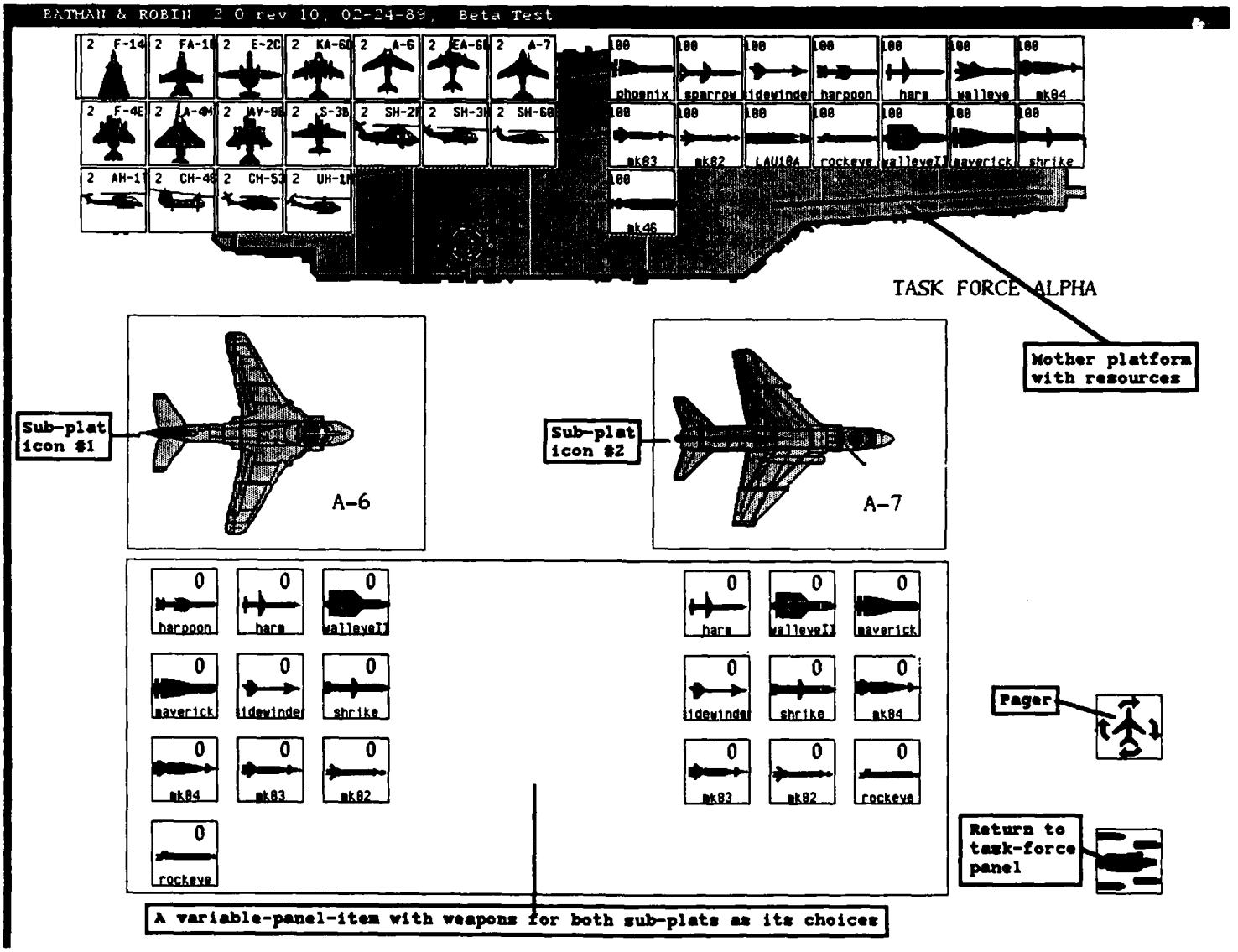


Figure 8. Blue Force Loadout Panel

3. A large icon of the *mother* platform with small icons for the *sub* platforms and weapons that exist on it.
4. An item that will hold the first *sub* platform being loaded out.
5. An item that will hold the second *sub* platform being loaded out.
6. A variable panel-choice item with icons of all the possible weapons that can be loaded out on either of the two *sub* platforms currently displayed.

Exported Functions:

start_subplat_loadout, *create_loadout_panel_items*, *display_resources_on_mothers_icon*, *notify_update_proj_count*, *set_proj_num*, *update_mothers_icon*, *load_next_plat_types*, *get_next_type_pair*, *load_projectile_records*, and *finished_loading_this_mother*.

10.3 Vector-Logic Grid Packages

To date, **grid** is the only package in this group.

10.3.1 Grid

This package is responsible for allowing the user to define the range of the vector-logic or defensive grid, and for updating the map and grid when the user chooses to zoom in and out. The specifications for the grid are as follows:

- The simulation plane is assumed to be a Cartesian coordinate system with Victor/Lima (V/L), the defended point, at the origin, (0,0). The grid is drawn with its center at V/L.
- The vector-logic grid is displayed for a full 360 degrees.
- A fixed-range increment for grid labels, i.e., 50 nautical mile tic marks.
- Angle or azimuth increments are expressed in integers, e.g., 15 degrees.

Refer to **World Database II** section below for more information on interacting with the maps.

Exported Functions:

display_grid_changes, *copy_map_to_display_pr*, and *draw_grid_on_pr*.

10.4 Deployment Packages

The packages in this group coordinate deploying Blue task forces in a chosen warfare theater, and setting initial alert levels for *sub* platforms. Figure 9 illustrates a typical deployment screen that will serve as a useful reference for the following subsections.

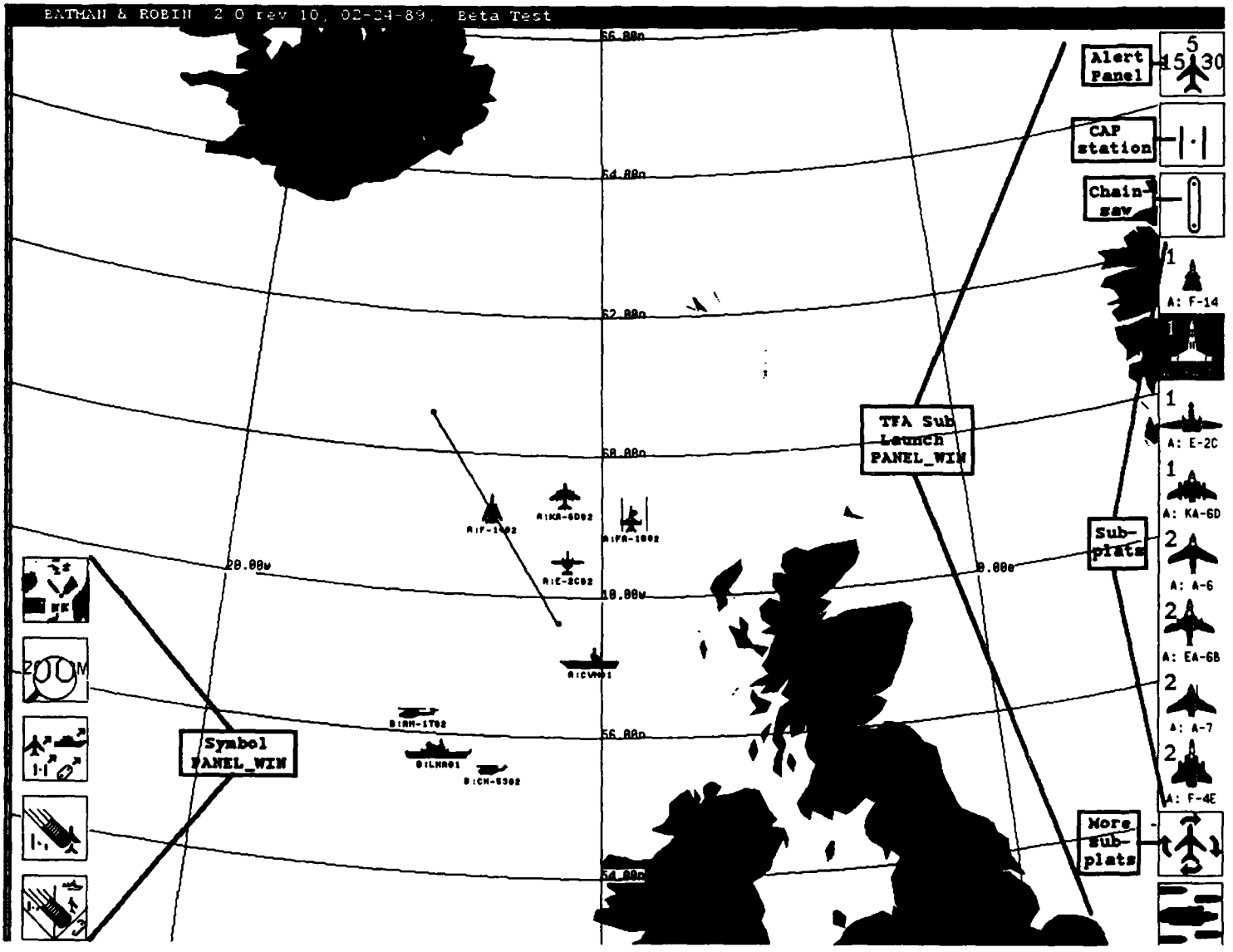


Figure 9. Deployment Screen

The deployment stage of the game is transitioned to by the *start_next_game_phase* routine of the **batman** package, mentioned earlier.

10.4.1 Cf_panels_create

This package creates the Console-force *PANEL_WIN*s described below. Figure 10 identifies the data structures where these *PANEL_WIN*s are defined. For more information on *PANEL_WIN*s, refer to the **Panel_win** subsection.

- The Symbol *PANEL_WIN* contains items for viewing the tactical-situation screen, zooming in or out, and moving and erasing platforms. This *PANEL_WIN* is placed flush to the left side of the deployment screen.
- The Task-Forces *PANEL_WIN* contains an icon for each Blue task force and the Red force hammer-and-sickle icon.
- The Simulation *PANEL_WIN*, which is displayed flush to the right side of the computer screen during BATMAN, contains the simulation clock, NTDS (Navy Tactical Data System) icons for viewing selected portions of the battle, i.e., air, surface, or subsurface warfare only, icons for making visible specified radar and sonar coverage of Blue force, the icon for accessing zooming and panning, and the icon for displaying the status windows of Blue air, surface, or subsurface platforms.
- *Sibling* Launch *PANEL_WIN*s, one for each Blue task force, contains an icon for each *sibling* platform in the task force and an icon that returns to the Task-Forces *PANEL_WIN*.
- *Sub* Launch *PANEL_WIN*s, one for each task force, contains an icon for each *sub* platform in the task force, an alert icon, a chainsaw icon, a CAP station icon, and an icon that returns to the Task-Forces *PANEL_WIN*.
- Alert *PANEL_WIN*s, one for each task force, contains a Ready 5, 15, and 30 icon for each *sub* platform in the task force. For more information, refer to the **Alert** subsection below.

While deploying tactical assets, all of the above but the Simulation *PANEL_WIN* are accessible. During the simulation, all of the above but the Symbol *PANEL_WIN*, the Task-Forces *PANEL_WIN*, and the *sibling* Launch *PANEL_WIN*s are accessible. Therefore, the *sub* Launch *PANEL_WIN*s and the *alert PANEL_WIN*s are used in both deploying assets and managing the battle.

Exported Functions:

init_symbol_icons, *create_symbol_panel*, *create_tf_panel_items*, *create_main_panel_items*, *display_plat_count*, *create_tf_launch_panel_items*, and *create_subplat_panel_items*.

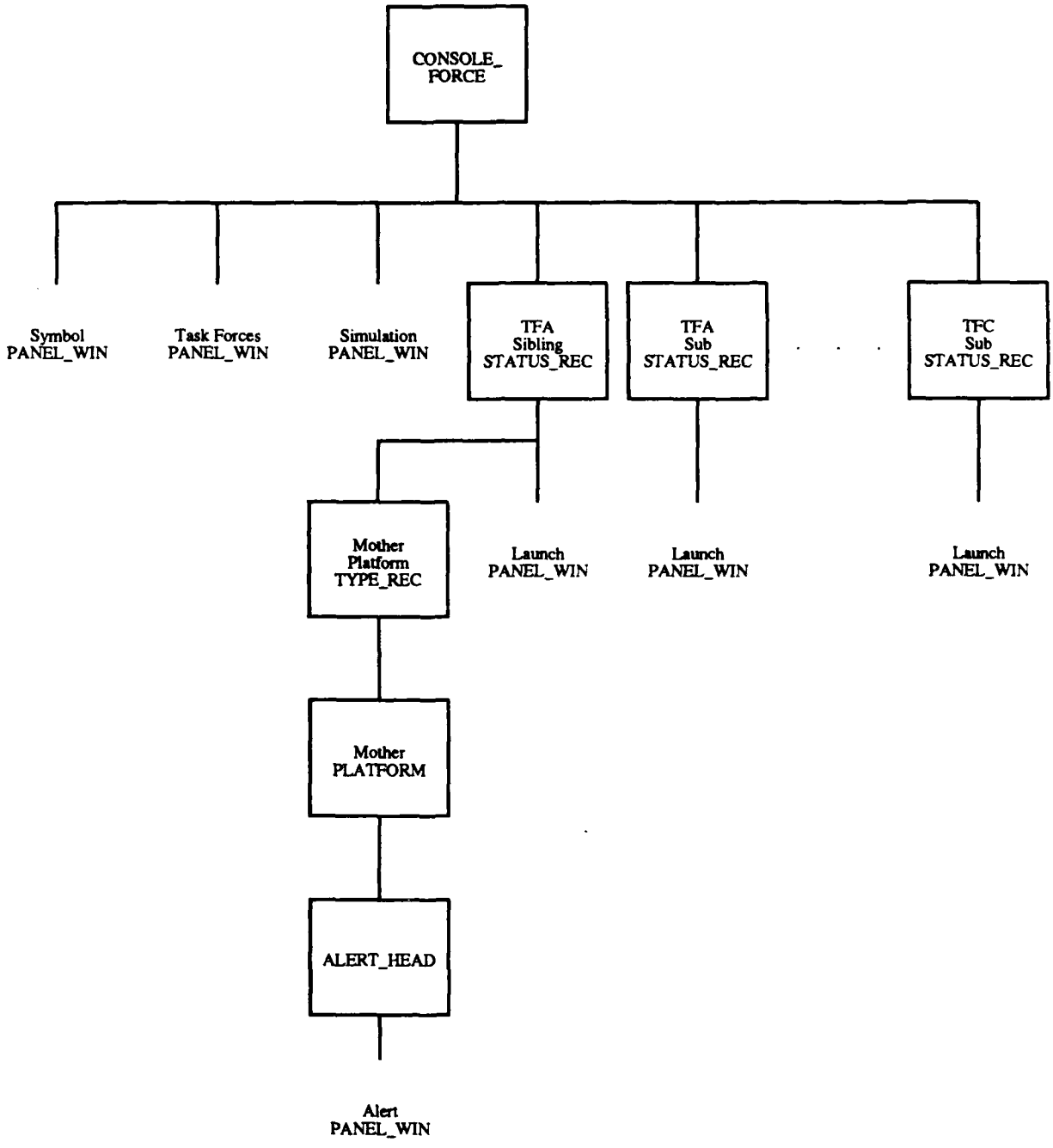


Figure 10. Deployment *PANEL_WIN*s

10.4.2 Cf_panels_notify

This package contains the Notify routines for the Console-force *PANEL_WIN*s created by the **cf_panels_create** package.

Exported Functions:

launch_panel_requested, *symbol_panel_notify*, *tf_panel_notify*, *main_status_notify*, *subplat_launch_notify*, *tf_plat_launch_notify*, *air_notify*, *subsurf_notify*, *surface_notify*, *air_radar_notify*, *surf_radar_notify* *threat_toggle_notify*, and *subsurf_radar_notify*.

10.4.3 Alert

This package contains routines to model alert or readiness states on *mother* or *home-base* platforms. Four alert states are available: Alert 5, Alert 15, Alert 30, and NULL Alert, i.e., a *sub* platform that is already launched. The minutes to launch time are not simulated.

An *ALERT_HEAD* structure (see *BATMAN & ROBIN Data Structures* section) is created and maintained for every *mother* platform that is in the Console force. The types of platforms in the Alert *PANEL_WIN* are duplicates of those found in the respective *sub* Launch *PANEL_WIN*. Platforms are moved among the readiness states with the function *find_plat_on_level*, which returns the first *sub* platform at the specified alert, and *put_plat_on_level*, which updates the *sub* platform's Alert state. Only *sub* platforms on Ready 5 can be launched. Platforms must be moved from lower to higher alert states for launching, e.g., moved from Ready 15 to Ready 5.

Exported Functions:

put_plat_on_level, *find_plat_on_level*, and *create_alert_panel_win*.

10.4.4 Symbol_manager

This package manages the placement, movement, and removal of *object* icons in the *maincanvas* (where the maps are drawn) during tactical deployment. *Object* icons include Console-force platform, CAP station, and chainsaw icons. The functions in this package are generally called when a mouse selection is made in the *maincanvas*, e.g., specifying the location to deploy an F-14. These calls have to occur within the right context. For example, the graphic object in Sub-Lunch Panel_Wins is selected prior to indicating with the mouse where this aircraft is to be placed within the *maincanvas* in order to initiate drawing the small F-14 icon in the designed position.

Exported Functions:

free_chain_list, *cap_requested*, *chain_requested*, *move_requested*, *remove_requested*, *placed_on_chain*, *placed_on_grid*, *clear_requested*, and *clear_all_symbols*.

10.4.5 Find_symbols

This package contains utilities for locating *object* icons in the *maincanvas* given a mouse-selected coordinate. They are generally used for move operations to determine if there is an *object* icon at the mouse-selected location. These utilities are used during the deployment and battle-management phases of the simulation.

Exported Functions:

find_plat, *closest_mother*, *find_mother*, *find_refueler*, *find_a_dot*, and *find_chain_dest*.

10.5 BATMAN Simulation Packages

The packages described in this section control and perform the BATMAN simulation.

10.5.1 Timer

This package provides "set-up", simulation, and fixed-interval timers. The "set-up" timer tracks the amount of time the user takes to loadout and deploy Blue forces. The simulation timer tracks the amount of time the user is engaged in battle. The interval timer, while used in other areas of BATMAN & ROBIN software, is described here since it plays such a critical role in driving the simulated battle.

The interval timer provided by SunView's notifier (SunView Programmer's Guide, 1988) will generate a software interrupt upon every lapse of a timer. It is based on the UNIX functions *setitimer* and *getitimer* (SunOS Reference Manual, 1988), but is only accessible through the notifier under the SunView programming environment. The interval timer is initialized with the notifier function *notify_set_itimer_func*. The caller specifies the duration between interrupts and an interrupt service routine.

BATMAN & ROBIN employ the interval timer to build a psuedo-UNIX kernel for running non-user-input processes while an individual interacts with the system. This kernel will be referred to as the Interval Timer Kernel (ITK), and is illustrated in Figure 11.

During the simulated battle, for example, the ITK can be processing a platform detection algorithm while the user is moving the mouse to make a selection.

The ITK consists of a list of *TIMER_NODE*s, and the function that operates on this list, *timer_node_updates*, which is the interval timer's interrupt service routine. The data-dictionary definition for a *TIMER_NODE* is as follows:

TIMER_NODE =
non-user-input-function-name +
function-data +
num-itimer-interrupts-to-wait-before-calling +
num-times-waited-so-far +
num-times-to-call-function +
pausable-flag +
LIST_NODE

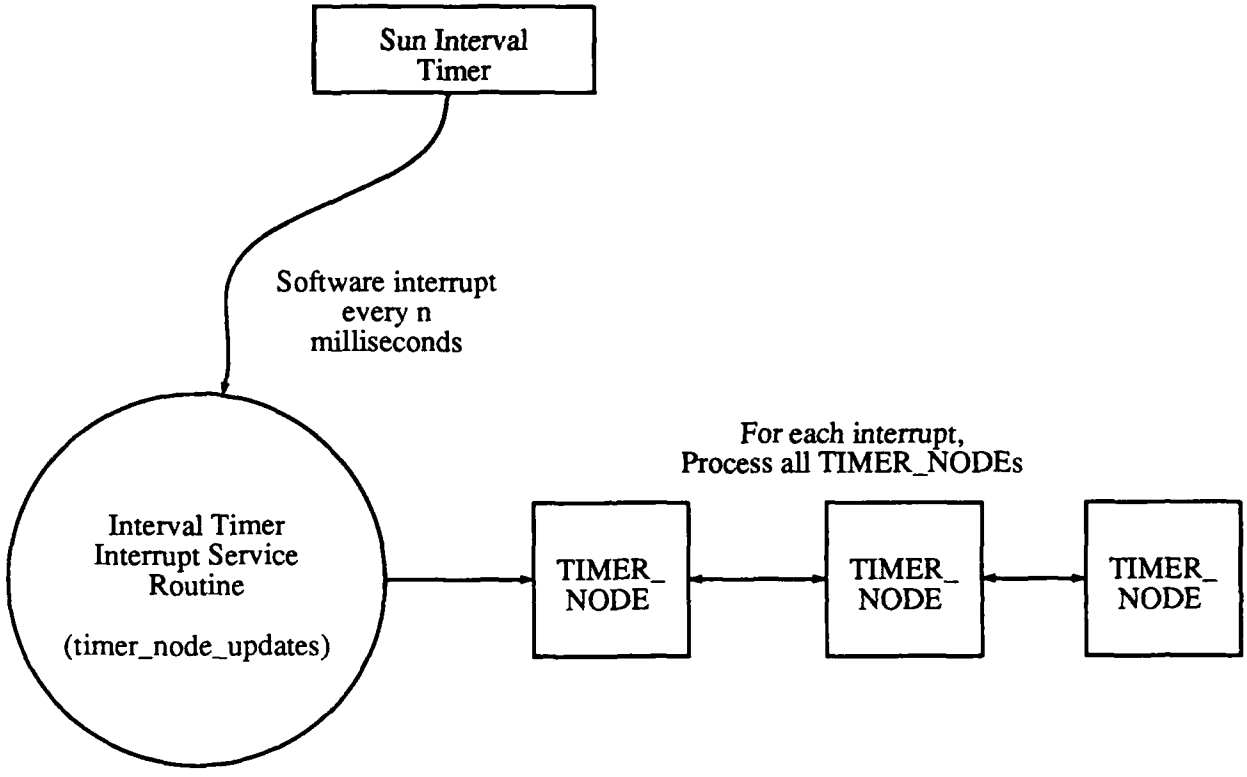


Figure 11. Interval Timer Kernel

The ITK will process every TIMER_NODE in the list for every interrupt generated by the interval timer. The high-level algorithm for *timer_node_updates* is as follows:

```
Begin
    If (not still servicing last interval timer interrupt)
        Get first TIMER_NODE from list
        While (not at end of TIMER_NODE list)
            If ((game is not currently paused) or
                    (pausable-flag is not set))
                Increment num-times-waited-so-far
                If (num-times-to-call-function has been reached)
                    Delete this TIMER_NODE from list
                Elseif (num-times-waited-so-far is greater than
                        num-itimer-interrupts-to-wait-before-calling)
                    Call non-user-input-function-name
```

```
                    Decrement num-times-to-call-function
                    Reset num-times-waited-so-far to zero
                Endif
            Endif
            Get next TIMER_NODE from list
        Endwhile
    Endif
End
```

Exported Functions:

set_the_notify_itimer, *timer_node_updates*, *add_to_the_interval_timer*, *remove_from_the_interval_timer*, *check_for_user_interrupts*, *start_setup_timer*, *stop_setup_timer*, *start_elapsed_timer*, *stop_elapsed_timer*, *set_time_values*, *calculate_time_stats*, *init_time_factor*, and *time_handler*.

10.5.2 Engine

This package contains the data structures and code that drives the BATMAN simulation. The controlling function is *simulation_engine* which operates from a circular list of *ENGINE_NODE*s. Figure 12 illustrates this list, and the following is the data-dictionary definition for an *ENGINE_NODE*.

```
ENGINE_NODE          =
        engine-node-type                    +
        function-name                       +
        function-data                       +
        LIST_NODE
```

where:

```
engine-node-type     =
        clock                               |
        map-copy                            |
        update-plat                         |
        red-detect-build                    |
        red-detect-plat                     |
        blue-detect-build                   |
        blue-detect-plat                    |
        draw-plat                           |
        update-misc                         |
        display-changes
```

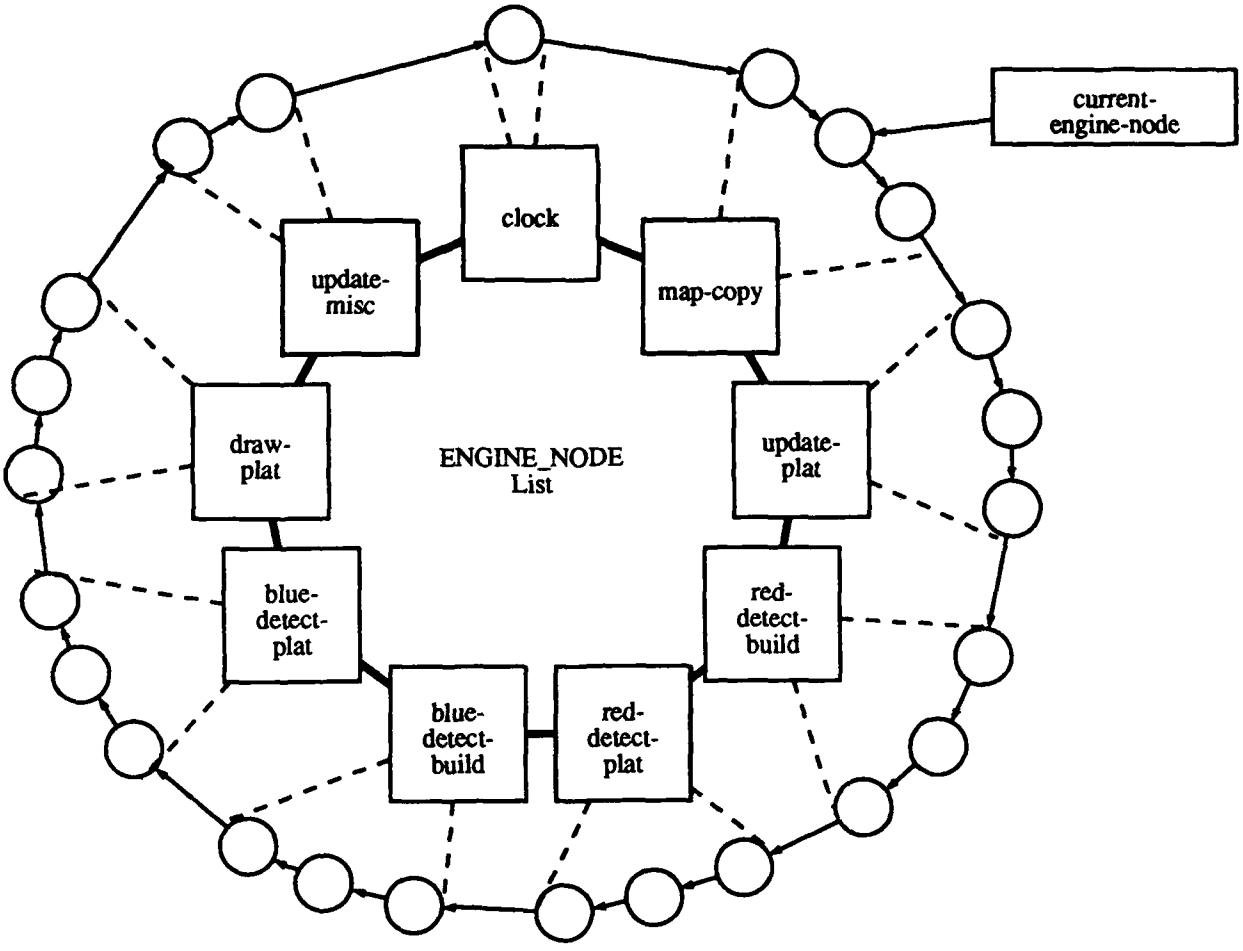


Figure 12. *ENGINE_NODE* List

*ENGINE_NODE*s are grouped into the above engine-node-types, and the list is maintained so that all nodes of the same engine-node-type are contiguous. For example, there are generally six *ENGINE_NODE*s of engine-node-type draw-plat: a Red and a Blue air, surface, and subsurface platform drawing *ENGINE_NODE*.

Before starting the battle simulation, the *ENGINE_NODE* list is initialized, and the "current-engine-node" is set to an arbitrary node. The initial state of the list is approximate to the illustration in Figure 12. Then, the simulation is started by adding a *TIMER_NODE* with *simulation_engine* as its function to the Interval Timer Kernel's list (see **Timer** subsection). Then, upon every software interrupt generated by the interval timer, the ITK will call *simulation_engine*. This continues until the simulation is terminated. *Simulation_engine* refers to its circular list of *ENGINE_NODE*s to determine which functions it should call.

Each call to *simulation_engine* is only allowed to run for the amount of time specified by the

msec_engine_update_interval parameter of the Object-Definition database (see Part III of this document). Imposing this time limit will periodically return control to the system so that it can process user input. Therefore, it may take a few calls of *simulation_engine* before one cycle of the *ENGINE_NODE* list is completed. The high-level algorithm for *simulation_engine* is as follows:

```
Begin
    Get the current time and start the engine timer
    While (forever)
        Call the function that resides in the current-engine-node
        Set current-engine-node to next node in list
        Get the current time
        If (time elapsed between start and current is
                greater than msec_engine_update_interval)
            Break the loop
        Endif
    Endwhile
End
```

Figure 13 illustrates the interaction between the interval timer and the simulation engine.

The most powerful engine attribute is that the *ENGINE_NODE* list it operates from can change during the simulation, dramatically increasing the engine's functionality. The *ENGINE_NODE* list changes depending on what the battle manager does. For example, if the battle manager requests that the outer-air battle not be visible, this is handled by removing the *ENGINE_NODE* that draws air platforms. If the battle manager requests that the outer-air becomes visible, the node is simply added back to the list.

Exported Functions:

simulation_engine, *check_for_simulation_termination*, *add_engine_node*, *remove_engine_node*, *set_confidence*, *simulation_input_handler*, and *setup_input_handler*.

10.5.3 Detect

This package contains functions that compute positional relationships between platforms and, using the detection-units aboard each of them, determines which Red threats a specified Blue platform can detect. Currently, the Willard-Leuker space tree model is used to perform these calculations. This model, as well as others, will be documented in *BATMAN & ROBIN: Human-Computer Interface and Simulation Models* (Federico, Bickel, Ullrich, & Bridges, in preparation). BATMAN & ROBIN can be adapted to use more complex detection models than Willard-Leuker.

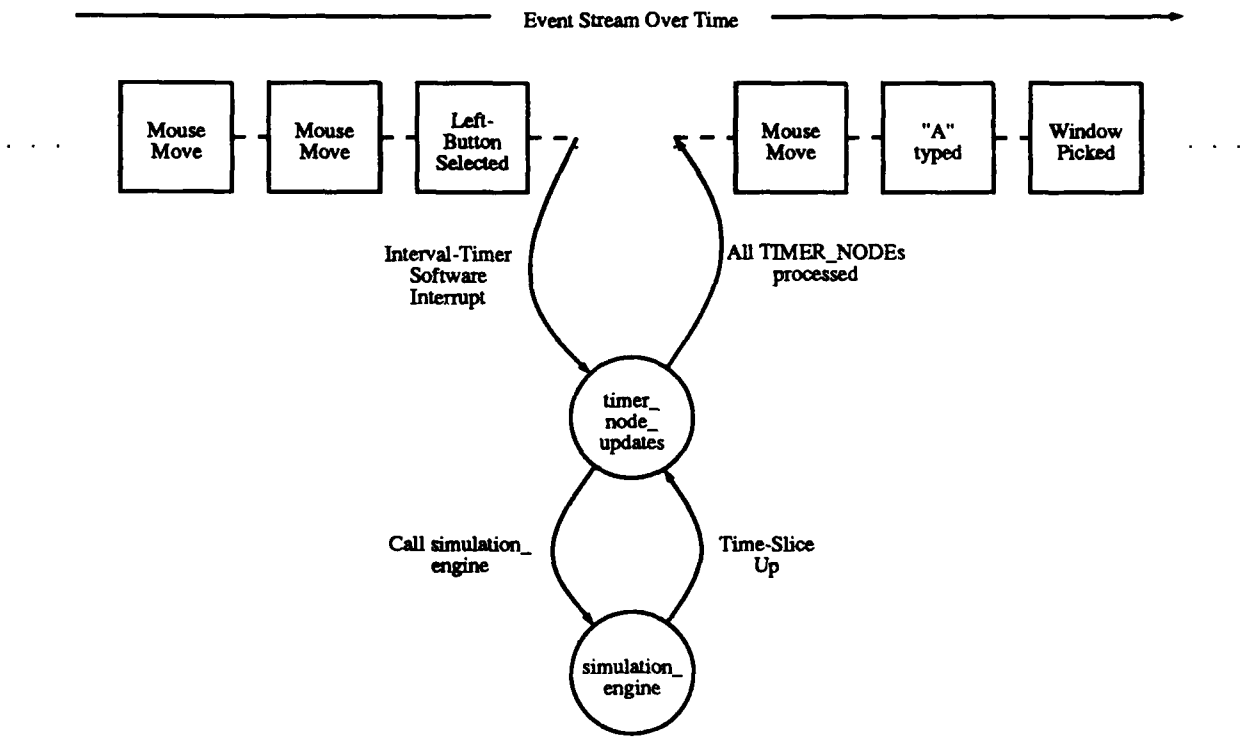


Figure 13. Interval Timer's Relation to the Simulation Engine

Each cycle of the simulation engine will build two Willard-Leuker space trees, one for Red platforms and one for Blue platforms. These are coordinated by "red-detect-build" and "blue-detect-build" *ENGINE_NODE*s, respectively (refer to **Engine** subsection above). After the Willard-Leuker space tree is built, Red platforms are detected for a specific Blue platform with the function *detect_plats*. The sensitivities of the platform's detection-units, which are defined in the Object-Definition Database (see Part III of this document), are the primary variables in determining which platform's are detected.

Exported Functions:

build_secdata, *free_secdata*, *build_space_tree*, and *detect_plats*.

10.5.4 Plat_detect_funcs

These functions use a platform's detection unit, e.g., radar or sonar, to locate hostile platforms, and simulate firing weapons at the detected raiders. These features are coordinated by "red-detect-plat" and "blue-detect-plat" *ENGINE_NODE*s (refer to **Engine** subsection above). Detecting threat platforms is accomplished by the *detect_plats* function from the **detect** package described above, and coordinated by routines in this package. Firing weapons at the detected platforms is also initiated by routines in this package.

Each detection unit on a platform is given a **detect_func** and a **weapons_release_func** in the Object-Definition Database (see Part III of this document) to control the unit's weapon-firing behavior. For example, a detection unit on a carrier to detect air threats would be assigned a **detect_func** for detecting hostile air threats and a **weapons_release_func** that would launch surface-to-air missiles at the raiders. The **detect_funcs** in this package are *detection*, *comjam_emission*, and *shoot_cvn_detection*. The **weapons_release_funcs** in this package are *fire_projectiles*, *fire_crz_missiles*, and *shoot_mother_detection*.

Exported Functions:

fire_projectiles, *fire_crz_missiles*, *comjam_emission*, *interceptor_detection*, *shoot_mother_detection*, *detection*, and *detect_platforms*,

10.5.5 Plat_draw_funcs

This package contains functions to draw platforms, CAP stations, chainsaws, radar and sonar coverage, chaff corridors, comjam emissions, antiship missiles, firing lines, and explosions in the *maincanvas* during the simulation. Hence, this package can be viewed as a toolbox of drawing functions that update the user's view of the battle. Drawing platform's is coordinated by "draw-plat" *ENGINE_NODE*s (refer to **Engine** subsection above). Other drawing functions may be called depending on what happens during the battle simulation. For example, an explosion is drawn when a platform is destroyed. Or, radar coverage for air platforms becomes visible if the user requests it. When the user alters his view of the battle, e.g., turning on surface-radar coverage, "update-misc" *ENGINE_NODE*s are added to the *ENGINE_NODE* list to depict the modified view.

Exported Functions:

draw_platforms, *draw_radar*, *draw_detection*, *draw_chaff*, *draw_missile*, *draw_plat_ao_cj*, *draw_plat_mb_jammed*, *explosion*, *draw_shot_line*, *draw_plat*, *draw_chain_sym*, and *init_chains_on_map*

10.5.6 Plat_list_funcs

This package contains routines to update a number of lists maintained during the simulation, e.g., a list of destroyed Blue platforms.

Exported Functions:

add_to_launched_missile_lists, *remove_from_not_launched_lists*, *remove_from_launched_lists*, *add_to_chain_list*, *remove_from_chain_lists*,

remove_from_draw_lists, *add_to_active_lists*, *remove_from_active_lists*, and *place_plat_in_dead_list*.

10.5.7 Plat_update_funcs

This package contains routines that control and update the movement and status of all Red and Blue platforms during the simulation. This includes routines called by user input, e.g., request to land an air platform on a carrier (*land_plat*), and by the simulation engine, e.g., update a platform's movement on a chainsaw (*follow_chain_path*). Each type of platform is assigned default update functions in the Object-Definition Database (refer to Part III of this document).

Exported Functions:

sink_ship, *splash_aircraft*, *set_plat_vector*, *null_func*, *land_platform*, *follow_chain_path*, *follow_tree_path*, *move_to_point*, *missile_move_to_point*, *land_plat*, *launch_missile*, *launch_plat*, *su_launch_to_chain*, *su_launch_to_point*, *move_to_land*, *move_to_refuel*, *hover*, *update_plats_states_and_set_stats*, *update_platforms*, *timed_update_platforms*, *timed_launch*, *interceptor_change_plat_direction*, *change_plat_direction*, and *set_one_plat_radar_flag*.

10.5.8 Status

This package contains functions to construct and display individual, *sibling*, and *sub* platform status windows. The *STATUS_WIN* structure, defined in this package, is used to implement these three types of status windows. A *STATUS_WIN* is on a Sun *Pixrect* (Sun Pixrect Reference Manual, 1988), but contains additional information to provide more functionality for BATMAN & ROBIN.

One *STATUS_WIN*, declared in the *CONSOLE_FORCE* structure (refer to **BATMAN & ROBIN Global Data Structures** above), is used to display the status of an individual platform. Only a single platform's status can be displayed at any one time. The individual status window is currently displayed in the lower left corner of the battle display.

Sibling platform status windows provide the status of all *sibling* platforms in the designated task force. *Sub* platform status windows provide the status of all *sub* platforms in the designated task force. A *STATUS_WIN* structure is declared in each *STATUS_REC* from the *CONSOLE_FORCE* structure to hold these windows. The *sibling* and *sub* platform status windows are currently displayed in the upper left corner of the battle screen.

The display of these three status windows is mutually exclusive, i.e., only one of them can be displayed at a time. When the user requests to view one of these status windows, an "update-misc" *ENGINE_NODE* is added to the *ENGINE_NODE* list to display the selected status window (refer to **Engine** subsection above). When the user no longer chooses to view the status window, the *ENGINE_NODE* is removed from the list.

Throughout BATMAN, Blue platform status is updated as follows: Each *PLATFORM* data

structure has a pointer to the force's individual *STATUS_WIN* and the *sibling* or *sub* platform *STATUS_WIN* to which it belongs. Then, during the battle, each active platform writes its updated status into both the individual *STATUS_WIN* and the appropriate "multiple-platform" *STATUS_WIN* (either *sibling* or *sub*). The updates are done in the "update-plat" stage of the simulation engine (refer to **plat_update_funcs** subsection above). The updates are only visible if a status window is displayed.

Exported Functions:

show_single_status, *hide_single_status*, *display_status*, *highlight_status_group*, *unhighlight_status_group*, *draw_plats_status_labels*, *create_indiv_status_window*, *create_status_windows*, and *destroy_status_win*.

10.6 Performance Measures Packages

The packages in this group compute and display objective, automatically recorded, multivariate performance measures (sometimes referred to as statistics) for the most recent BATMAN scenario. Performance measures can be viewed at the console, sent to a printer, or written to a file. Figure 14 provides an example of a performance measures display screen.

The header file **stats_recs.h** declares four data structures used by the performance measures packages. Their names and purposes are listed in Table 2.

Table 2.

Performance Measures Data Structures

Data Structure	Purpose
STATS_MANAGER_WIN	Performance Measures Manager *PANEL_WIN*
STATISTICS_WIN	Performance Measures *PANEL_WIN*
STATS_FIELDS	Hold data collected during simulation
STATS_HEAD	Ties above three structures together

STATS_MANAGER_WIN provides a *PANEL_WIN* that allows the user to select the types of platforms (air, surface, or subsurface) and point of view (Red or Blue) of the performance measures display. *STATISTICS_WIN* provides a *PANEL_WIN* to hold the performance measures when they are displayed. One of its Panel Items is a large *Pixrect* where the performance measures are written. *STATS_FIELDS* holds the data collected during the simulated battle that will be used to compute the performance measures. *STATS_HEAD* brings the above three structures together in one place, with some other information global to the performance measures packages.

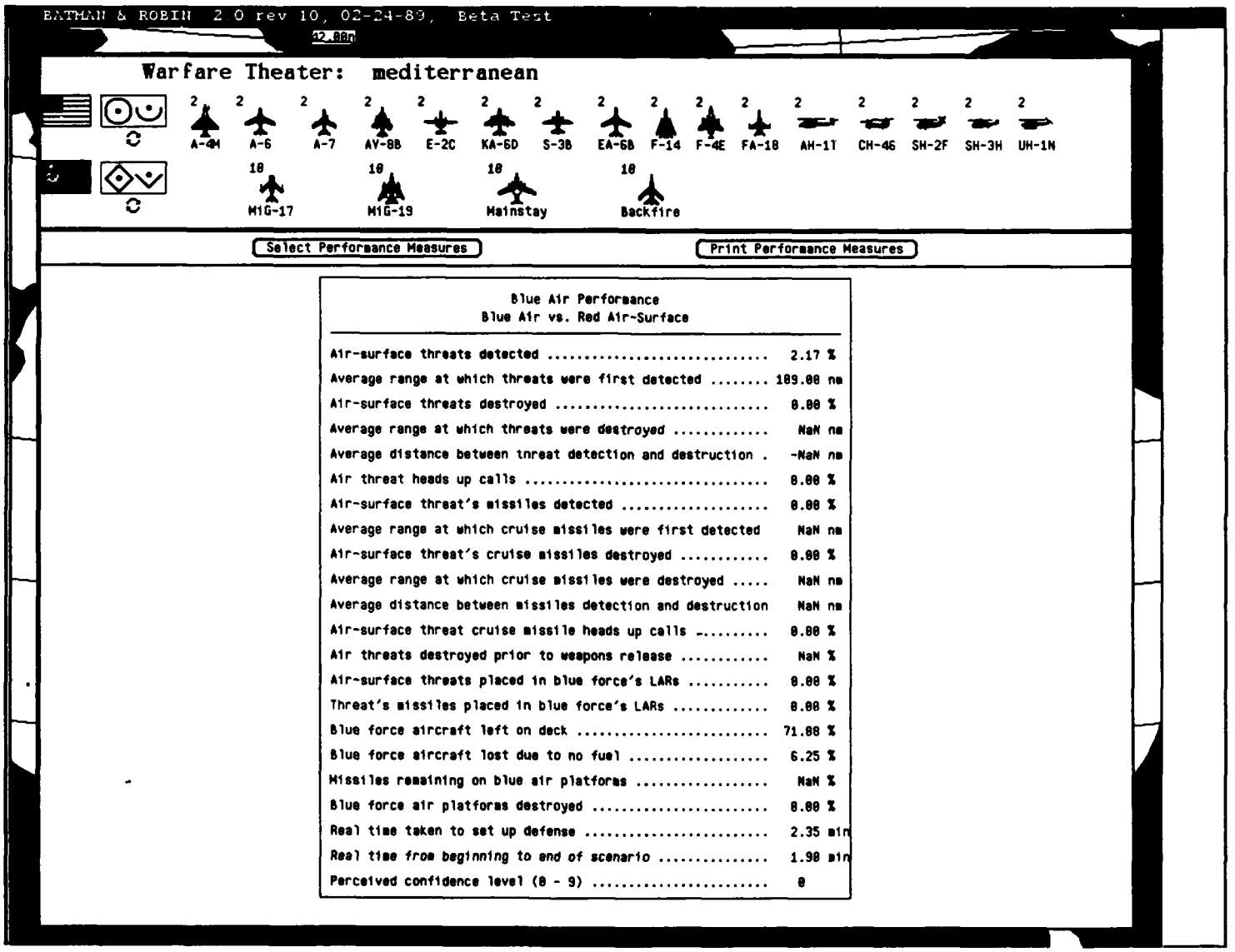


Figure 14. Performance Measures Screen

10.6.1 Stats

This package contains functions that initialize the *STATS_HEAD* structure, create the *PANEL_WIN*s in the *STATS_MANAGER_WIN* and *STATISTICS_WIN* structures, and display or print performance measures.

Exported Functions:

create_stats_head, *write_blue_stats*, *write_red_stats*, *print_blue_stats*, *print_red_stats*, *display_the_stats*, *clear_all_stats*, and *free_stats_head*.

10.6.2 Stats_compute_funcs

The functions in this package use the data collected during the simulated battle to compute the performance measures. The *STATS_FIELDS* structure holds the collected data. These functions can be used to calculate the performance measures from either the Red or Blue Force's point of view. In general, one function is dedicated to calculating each performance measure.

Exported Functions:

compute_threat_core, *compute_threat_headsup*, *compute_missile_core*, *compute_missile_headsup*, *compute_destroyed_weapons_release*, *compute_threat_in_lar*, *compute_left_on_deck*, *compute_lost_to_no_fuel*, *compute_missiles_remaining*, *compute_plat_destroyed*, *compute_missile_launch*, *compute_avg_launch_range*, *compute_targets_destroyed*, and *divide*.

10.6.3 Stats_notify

This package contains the Notify functions for the *PANEL_WIN*s from the *STATS_MANAGER_WIN* and *STATISTICS_WIN* structures.

Exported Functions:

exit_stats_notify, *return_manager_notify*, *print_stats_notify*, *show_blue_stats_notify*, *show_red_stats_notify*, *show_both_stats_notify*, *blue_air_notify*, *blue_surf_notify*, *blue_sub_notify*, *red_air_notify*, *red_surf_notify*, and *red_sub_notify*.

10.6.4 Stats_update_funcs

This package contains functions that the simulation uses to send battle-related data to the *STATS_FIELDS* structure, thereby recording a history of the battle as it occurs. Many of the functions from the simulation engine (refer to **BATMAN Simulation** section above) make periodic calls to these functions.

Exported Functions:

stats_load_user_info, *stats_load_plat_info*, *stats_load_confidence_level*,

stats_load_setup_time, *stats_load_scen_time*, *stats_load_num_of_plats*, *stats_load_num_of_targets*, *stats_plat_detected*, *stats_plat_destroyed*, *stats_missile_detected*, *stats_missile_destroyed*, *stats_target_destroyed*, *stats_plat_headsup*, *stats_missile_headsup*, *stats_destroyed_weapons_release*, *stats_plats_in_lar*, *stats_missiles_in_lar*, *stats_out_of_fuel*, *stats_plat_launch*, and *stats_missile_launch*.

10.6.5 Stats_verify

This package contains functions that write performance measures to a file in the user's directory. For more information on the User Database, refer to Part III of this document.

Exported Functions:

stats_print, *init_stats_debug*, and *close_stats_debug*.

10.7 ROBIN Packages

The packages in this group implement ROBIN, the system's scenario generator. They are responsible for providing a friendly interface to creating, editing, and viewing raids for different warfare theaters as well as assigning tactical assets to Blue task forces.

Figure 15 provides a map between the features provided by ROBIN, and the primary package providing that feature.

The header file **robin_globals.h** contains global variable used by all ROBIN packages.

10.7.1 Robin_init

This package contains functions that initialize ROBIN's global variables, control the creation of ROBIN's windows and panels prior to a session, and remove ROBIN's windows and panels after a session.

Exported Functions:

create_robin_windows, *init_robin_data*, and *end_robin_session*.

10.7.2 Robin_manage

This package controls ROBIN's administrative features including browsing scenarios, saving scenarios to disk, deleting scenarios, and transitioning to the scenario editor. It also contains a function that creates the manager *PANEL_WIN*.

Exported Functions:

display_theater, *redisplay*, *display_manager_panel*, *new_passwd_write*, and *init_manager_panel*.

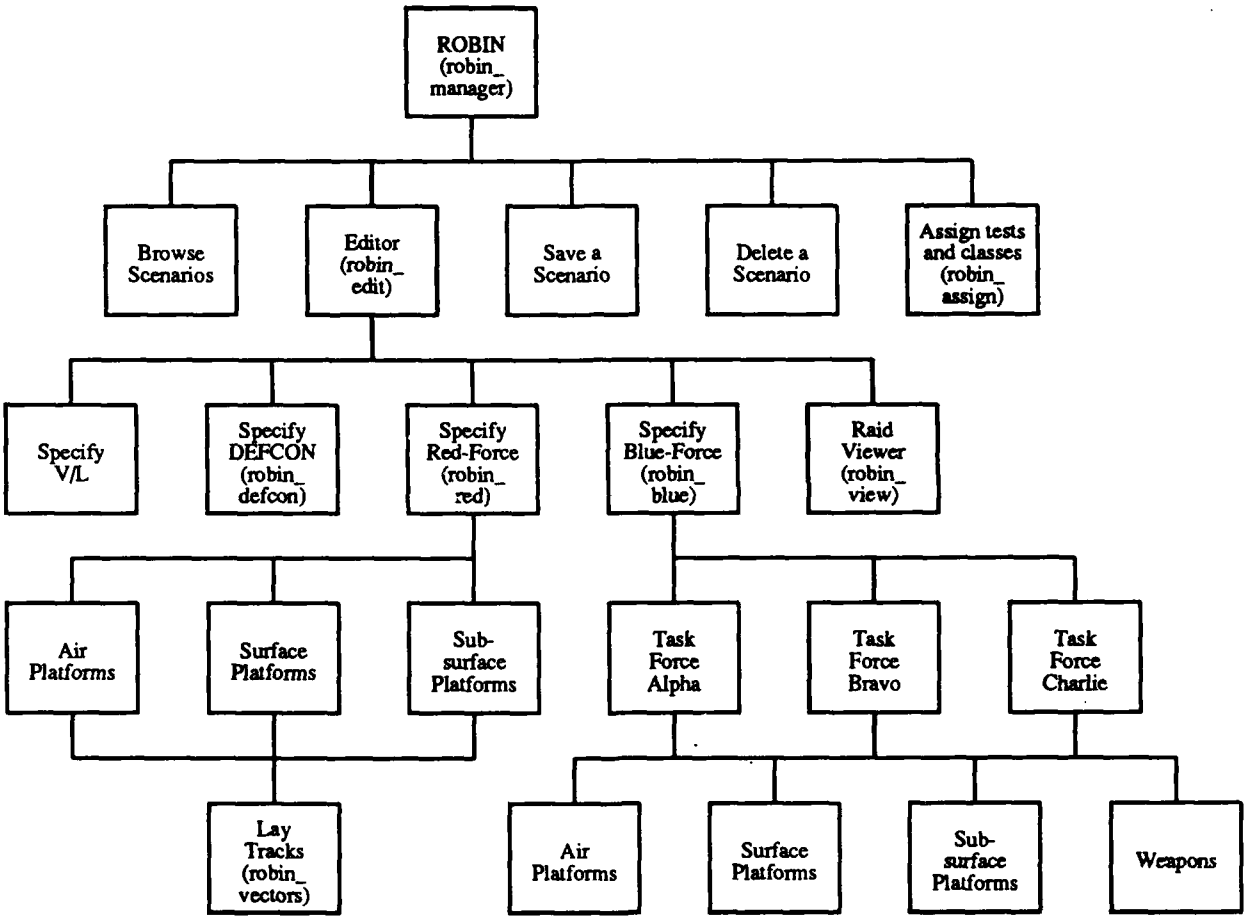


Figure 15. ROBIN Features to Package Map

10.7.3 Robin_assign

This package provides routines that will assign in ROBIN specified sets of scenarios to particular groups of students or system users. That is, scenarios or test items can be clustered into sets or tests, and users or students are placed into groups or classes. Within a set, scenarios can be administered either randomly or in a specified order to a group of BATMAN users.

Exported Functions:

init_assignment_panel, read_assignment_data, write_asssignment_data,

10.7.4 Robin_edit

This package provides controlling routines for the scenario editor. It creates the editor *PANEL_WIN* and regulates transitioning to specific edit features such as placing the defended point, V/L, specifying the Red-force raid, or allocating Blue-force assets. Only one scenario may be edited at a time.

Exported Functions:

display_editor_panel, redisplay_editor_panel, and *init_editor_panel.*

10.7.5 Robin_map

This package provides routines that interface with the *World Database II* maps, and are used by both the **robin_manage** and **robin_edit** packages. For more information on the maps, refer to the **World Database II** section below. Specific features provided by this package include displaying all warfare theaters, interactions with the miniature world map that appears in the upper left corner of the ROBIN manager, drawing the scale that borders the warfare theaters in the ROBIN editor, and placing and drawing V/L within a warfare theater.

Exported Functions:

scale, map_copy, notify_threat, notify_vl, notify_comjam_size, init_scenario_location, free_sm_map_prs, and *map_selected.*

10.7.6 Robin_blue

This package contains the Panel creation and notify routines that allow the user to specify the Blue force's tactical assets. The user can define up to three task forces designated TFA, TFB, and TFC. For each task force, the user indicates the air, surface, and subsurface platforms and weapons loadout for the scenario. In BATMAN, these will become the Blue force tactical assets that the battle manager has to allocate, deploy, and control.

Exported Functions:

notify_carrier_plane_done, notify_carrier_missile_done, notify_display_carrier_missiles, notify_display_carrier_planes, notify_blue_carrier,

notify_blue_done, *notify_ship*, *notify_plane*, *notify_projectile*, *display_blue_ships*, *init_blue_mp_panel*, *display_blue*, *display_plane_input*, *display_ship_input*, *display_proj_input*, *set_plane_on_carrier*, *set_missile_on_carrier*, *init_blue_ship_panel*, *init_sub_plat_panel*, and *init_missile_panel*.

10.7.7 Robin_red

This package contains the Panel creation and notify routines that permit the user to specify the Red-force attack for its air, surface, and subsurface platforms. Currently, Red air platforms are divided into fighters and bombers. This package works closely with the **robin_vectors** package described below.

Exported Functions:

notify_red_done, *set_chosen_item*, *get_new_swarm*, *set_new_swarm*, *set_red_menu*, *set_lat_lon_items*, *set_vector_changes*, *set_red_buttons*, *notify_erase*, *notify_eraseall*, *notify_hide*, *notify_red_panel*, *display_red_bombers*, *display_red_fighters*, *display_red_ships*, *display_red_subs*, *display_red_panel*, *toggle_robin_canvas_select*, *notify_main_menu*, and *init_red_panel*.

10.7.8 Robin_vectors

This package contains routines for placing Red-force tracks including laying a new track, extending an existing track, erasing defined tracks, hiding tracks from view, and laying chaff and comjam on tracks. This package works closely with the **robin_red** package above.

Exported Functions:

set_drawing_off, *destroy_draw_util_prs*, *init_draw_util_prs*, *get_list_head*, *set_list_head*, *find_node*, *add_vector*, *clear_path_force_values*, *clear_all_scenario_values*, *draw_paths*, *hide_track*, *all_nodes_invisible*, *all_nodes_visible*, *draw_plane*, *copy_objct*, *remove_node*, *vector_selected*, and *comjam_chaff_selected*.

10.7.9 Robin_view

This package provides routines that will preview the Red-force raid showing how these hostile platforms will maneuver during BATMAN and where they will lay chaff corridors or jam communications. In order to allow concurrent user input during previews, the Interval Timer (described in the **BATMAN Simulation** section) is used to divide processor time between the viewer and user input.

Exported Functions:

notify_compression, *notify_view_done*, and *notify_view*.

10.7.10Robin_defcon

This package is used to specify the DEFense CONdition (DEFCON) for the scenario. It contains Panel create and notify routines for the available DEFCONs.

Exported Functions:

display_defcon.

10.7.11Robin_io

This package contains functions that interface with the Scenario Database. For more information on the Scenario Database, refer to Part III of this document.

Exported Functions:

write_blue, *get_parent*, *string_to_type_recs*, *string_to_node*, *string_to_vector*, *file_write*, and *file_read.*

10.8 World Database II Packages

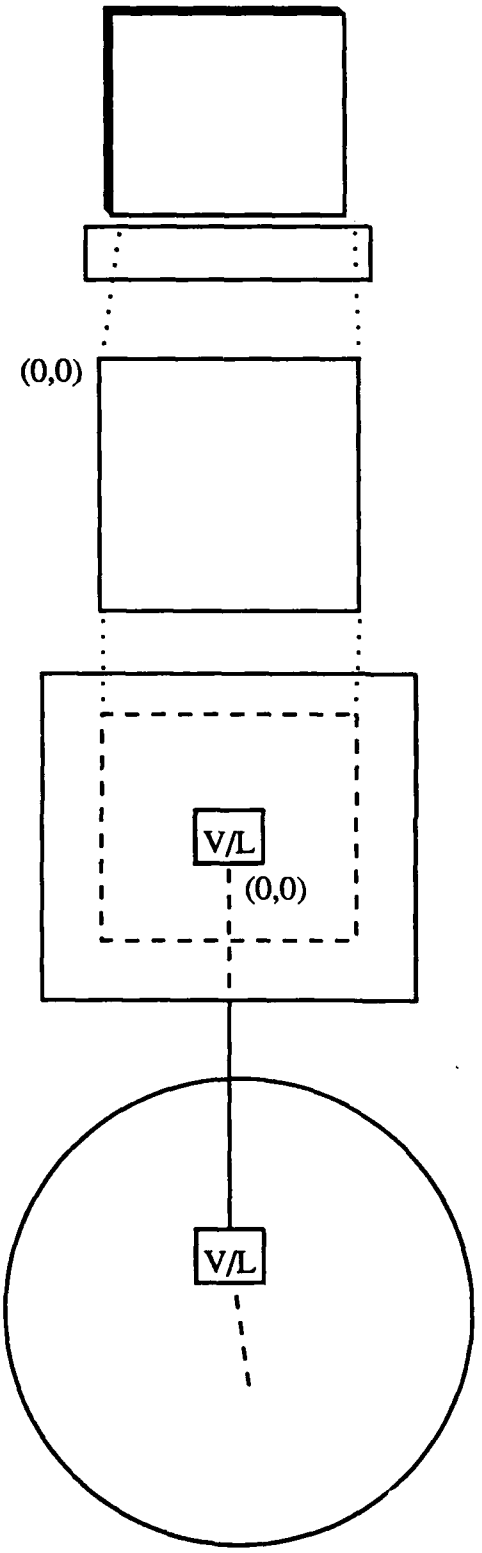
The package in this group, **mapdb**, provides an interface to the *World Database II* maps developed by the Applied Physics Laboratory of Johns Hopkins University. The specifications of these maps are as follows:

- The maps are rendered by an orthogonal projection of the globe onto a plane tangent from a specified latitude-longitude coordinate, and viewed from a specified range.
- The resolution of the views range from 16 to 2048 miles in diameter.
- Land masses are drawn with a polygon filling algorithm that draws the largest possible polygon first, and then fills in the details by drawing smaller and smaller polygons.
- The maps can be drawn with or without country borders.
- The maps can be drawn with or without latitude-longitude lines.

This document does not describe the map database or the software that controls it, only BATMAN & ROBIN's interface to it, i.e., **mapdb**.

In interacting with the *World Database II* maps, BATMAN & ROBIN use the coordinate systems described below and illustrated in Figure 16.

- The Global coordinate system is based on latitude and longitude. In ROBIN, all raid information will be stored in this system as well as the center point of the battle area.



Display Monitor - Map Canvas is drawn to BATMAN & ROBIN's window

MAP CANVAS - Origin is in upper left corner

MAP/SIMULATION PLANE - Origin is at Victor/Lima

GLOBAL - Victor/Lima is center point of battle

Figure 16. BATMAN & ROBIN Coordinate Systems

- The Map Plane is a Cartesian coordinate system tangent to the globe at the center point of the battle area with Y axis *parallel* to the longitude line of the center point. The map is an orthogonal projection from the globe to this plane. X and Y distances are in miles from the center of the earth along a plane through the center, not along the surface. The tangent point of this plane will not change during the simulation.

- The Simulation Plane is a Cartesian coordinate system with origin at the center of the battle area. X and Y are in increments of .1 miles. The distance in miles is linear on this coordinate system. This coordinate system is used for detection and calculation of all platform movements during the simulation. Platform locations will be stored in this coordinate system during the simulation and will be converted to the other systems when needed.

- The Map Canvas is the coordinate system of the *maincanvas* containing the visible portion of the Map Plane. The center of this coordinate system corresponds to an offset from the origin of the Map Plane. This coordinate system is used to draw all objects on the *maincanvas*, i.e., the battle area.

10.8.1 Mapdb

This package provides an interface to the *World Database II* maps. The routine *draw_map_on_pr* will draw a map onto a *Pixrect* given a latitude-longitude coordinate, a range, and boolean flags that indicate whether to include country borders and latitude-longitude lines. The other routines exported by this package convert locations between the above coordinate systems.

Exported Functions:

draw_map_on_pr, *mp_to_mc*, *mc_to_mp*, *mp_to_sp*, and *sp_to_mp*.

10.9 Tool Packages

10.9.1 Canvas_win

This package provides routines for creating and manipulating *CANVAS_WIN*s. A *CANVAS_WIN* is based upon a SunView Canvas, but contains additional information to provide more functionality. Its abstract data-dictionary definition is as follows:

CANVAS_WIN =
SunView Canvas +
visible-and-hidden-coordinates

A powerful feature of this package is that it provides a mechanism whereby a Canvas can contain pseudo Panel-button items.

Exported Functions:

get_canvas_win, *put_canvas_win*, *hide_canvas_win*, *show_canvas_win*, and

canvas_win_event_proc.

10.9.2 Colors

This package provides functions that initialize and maintain BATMAN & ROBIN's colormap which resides on disk in the file specified by the **colormap** parameter of the Object-Definition Database (see Part III of this document). The SunView routines *pr_putcolormap* and *pw_putcolormap* use this colormap to set-up its color palette.

Exported Functions:

set_default_window_colors, *set_pw_colors*, *set_pr_colors*, *make_colors*, *show_colors*, and *str_to_color*,

10.9.3 Event_ctrl (Playback)

This package provides an interposed event function, playback functions, and other miscellaneous event-related functions.

The interposed event function (SunView Programmer's Guide, Chapter 17 - Notifier, 1988) provides capabilities to monitor all events as they occur and react to certain events in non-standard ways depending on the application. Hence, it provides the programmer with some flexibility in the way events are handled. BATMAN & ROBIN use this interposition feature to screen for special interrupt events, e.g., ctrl-d, which dumps the current screen image to a printer or disk.

Playback is a feature that initially records an user's actions during a BATMAN scenario, and subsequently and programmatically recreates the individual's actions during the management of the battle. This is accomplished by recording all user events that occur during allocating, deploying, and managing tactical assets, and then regenerating those actions during playback.

Exported Functions:

init_playback_play, *generate_playback_interrupts*, *add_to_input_grabbers*, *remove_from_input_grabbers*, *add_event_ctrl_window*, *remove_event_ctrl_window*, *save_playback_event*, *show_window*, *hide_window*, and *switch_to_sim_mode*.

10.9.4 List_manager

This package provides standardized data structures and functions for manipulating doubly-linked lists. Each *LIST_NODE* contains previous and next pointers as well as a pointer to some generic data:

```
LIST_NODE                =
        ptr-to-previous_node                    +
        ptr-to-generic_data                     +
        ptr-to-next_node
```

LIST_HEAD =
first-node-in-list +
last-node-in-list

Exported Functions:

list_insert, *list_delete*, *list_next*, *list_add*, and *list_first*.

10.9.5 Number_pad

This package provides a pop-up number pad that is used for inputting integers into BATMAN & ROBIN, e.g., specifying the number of Phoenix missiles to load out on each F-14. The number pad is implemented with a *PANEL_WIN* where each digit (0 - 9), the backspace key, the enter key, and the display area are Panel items. If the enter key is selected and the number in the display area is within a specified range, the number is "accepted" and the number pad is automatically cleared from the screen.

Exported Functions:

init_pad; *get_pad_height*, *get_pad_width*, and *set_pad*.

10.9.6 Panel_win

This package provides routines for creating and manipulating *PANEL_WIN*s. A *PANEL_WIN* is based on a SunView Panel, but contains additional information to provide more functionality. Its abstract data-dictionary definition is as follows:

PANEL_WIN =
SunView Panel +
visible-and-hidden-coordinates +
0 {SunView Panel-items} N-1 +
number-of-items +
0 {icon-pixrect} N-1 +
0 {x-location-of-icon} N-1 +
0 {y-location-of-icon} N-1 +
number-of-icons

A powerful feature of this package is that it can create a SunView Panel-choice item with a variable number of options where each is labeled with an icon. For example, this feature is used during weapons loadout (refer to figure 8 in the **loadout** subsection). The number of different types of weapons that can be loaded out on an aircraft varies depending on the specific platform. The **panel_win** package can be used to handle these situations without requiring a different Panel-choice item for each air platform.

Exported Functions:

get_panel_win, *put_panel_win*, *hide_panel_win*, *show_panel_win*, *create_variable_panel_item*, *set_variable_panel_items_icons*, *down_click_event_proc*, and *colored_button_image*.

10.9.7 Popup_panel

This package provides a mechanism for displaying "pop-up" (overlaid) messages, which may contain button and/or a password-request items. It can be used to display general or error messages with a continuation button, "pseudo menus" where each option is a button, or a simple password request.

One private *PANEL_WIN* is used to store and display the popup messages. Upon each request to display a message, the local *PANEL_WIN* is cleared and then set-up with the specified attributes of the message. The "varargs" parameter passing mechanism is used to specify a varying attribute list that defines the appearance of the message panel. Defaults are used when required attributes are not specified.

Exported Functions:

init_popup, *popup_panel*, *hide_popup_panel*, *compare_passwd*, *get_popup_width*, and *get_popup_height*.

10.9.8 Scen_display

This package creates a Panel that displays a graphical summary of a scenario including the warfare area as well as Red-force and Blue-force tactical assets. It is used in ROBIN by the scenario browser and in BATMAN by the performance-measures packages.

Exported Functions:

load_plat_pic, *init_scen_display*, *create_scen_display_window*, *create_scen_display_head*, *clear_scen_display*, *free_scen_display*, *show_scen_display_win*, and *hide_scen_display_win*,

10.9.9 Utilities

This package contains utility routines for trigonometric calculations, database input and output, memory allocation, image drawing, and several other miscellaneous functions. These functions are used throughout the BATMAN & ROBIN software.

Exported Functions:

d_get_i, *d_get_s*, *arctan*, *hypotenuse*, *distance*, *error_message*, *nexttok*, *one_fscanf*, *add_to_pixrect_buffer*, *create_pixrect*, *create_alloc*, *set_preserved_mem*, *free_mem*, *get_font*, *get_picture*, *flip_picture*, *fix_string*, *write_string*, *write_label*, *pr_duplicate*, *free_alloc*, *free_pixrect*, *display_canvas_image*, *pr_vecbox*, *pw_vecbox*,

10.9.10Zoom

This package handles the zoom feature available in ROBIN while defining or editing a scenario and BATMAN while deploying or managing tactical assets. When activated, it provides a special mouse cursor and input handler to inform the user that the zoom feature is engaged. Then, when a new range is entered, the **mapdb** package (see above) is used to

reset and draw the new view.

Exported Functions:

redraw_forces, *zoom_input_handler*, and *zoom_notify*.

Part III: Database Descriptions

11.0 Purpose and Scope

This part of the document describes the BATMAN & ROBIN databases. It is intended primarily for software-maintenance personnel. However, those individuals with a technical background who want to understand the organization of the databases may also benefit from reading this part of the documentation. **At this time, the databases described herein are completely unclassified and initially intended for development and evaluation purposes only. These databases which are independent of the simulation software can be made classified if so desired.** The following sections describe the Object-Definition, Graphics, Scenario, and User Databases.

12.0 Object-Definition Database

The Object-Definition Database contains operational attributes of air, surface, and subsurface platforms, sensors, weapons, and miscellaneous system-configuration parameters. BATMAN & ROBIN were designed and developed to be generic and adaptable. That is, platforms, weapons, or system-configuration parameters may be easily added, deleted, or modified without changing source code. Relevant parameters are stored in a standard text file that can be altered with any text editor, e.g., **vi**. (It is intended to develop a natural language or direct-manipulation interface for interacting with this database.) This text file is referred to as the Object-Definition Database, and can be either classified or unclassified depending upon the user's needs.

12.1 Platforms and Weapons

In BATMAN & ROBIN an object-oriented programming approach was adopted to a feasible extent to represent platforms and weapons. A platform, e.g., F-14 or Kidd, has the following characteristics: it can move dynamically during BATMAN, detect other platforms, fire weapons, and experience battle damage (Federico, Bickel, Ullrich & Bridges, in preparation). There are also "restricted" platforms in this database, e.g., AS-4, which move dynamically, experience damage, but cannot carry or fire weapons. Each weapon has an associated "probability of kill" (Pk), which specifies the likelihood that it will destroy its target.

12.2 Database Attributes

The text file that contains the Object-Definition database must be named ".defaults", and must reside in the user's home UNIX directory. BATMAN & ROBIN read values that represent attributes, characteristics, or parameters of platforms, sensors, and weapons from

the **.defaults** file with Sun's "defaultstool" utility (SunView Systems Programmer's Guide, 1988). Each line in the **.defaults** file consists of an object, parameter, and value, formatted in the following way:

/object/parameter **"value"**

An **object** specifies the name of a platform, weapon, sensor, or system parameter, e.g., F-14. A **parameter** represents an attribute of an object, e.g., altitude. A **value** assigns a particular number or string to a parameter, e.g., 30,000 feet. The value must be enclosed by quotation marks. For example, the following line in the .defaults file would tell BATMAN & ROBIN that F-14s should fly at 30,000 feet:

/F-14/altitude "30000"

The following **.defaults** entries provide further elucidation of how objects are defined in BATMAN & ROBIN:

/F-14/class "platform"

/F-14/force_id "blue"

/F-14/stats_type "air"

/F-14/long_name "Tomcat"

/F-14/altitude "30000"

These values define the "F-14" as a Blue air platform flying at 30,000 feet, which can be referred to as "Tomcat".

The Object-Definition database is divided into six sections: platforms, weapons, sensors, system-configuration, performance-measures, and users. Although the order of these sections in the **.defaults** file is not relevant. It is recommended that BATMAN & ROBIN users adhere to this convention.

12.2.1 Platform Parameters

Not all attributes need to be defined for every object, e.g, it is assumed that fuel-consumption characteristics are inappropriate for nuclear powered ships. The following specify the possible platform parameters and acceptable values.

altitude, altitude_max and **altitude_min**: a platform's default, upper, and lower bounds in feet above or below sea level.

> ***value***: air platforms greater than zero, surface platforms zero, and subsurface platforms less than zero.

as_missiles: the type and number of AS (AntiShip) missiles that a threat-air platform carries.

value: a list of missiles separated by spaces where each missile is prefixed by a positive integer which indicates how many of a certain type. If the missile is not prefixed by an integer, then the number defaults to one. The following example allocates two AS-4s and one AS-5 to the "Backfire" aircraft:

/backfire/as_missiles "2 as-4 as-5"

change_direction_func: the name of the C function called when the user moves a platform.

value: *interceptor_change_plat_direction* for F-14s or F/A-18s; *change_plat_direction* for other platforms.

class: specifies if the object is a platform or weapon.

value: platform.

draw_func: the name of the C function called to draw a platform's icon.

value: *draw_plat_mb_jammed* will not draw a Blue platform when it is jammed; *draw_plat_ao_cj* will not draw a Red platform unless it has been detected; and *draw_missile* will draw "restricted platforms".

extra_fuel: the amount of *give* fuel in pounds a tanker aircraft, e.g., KA-6D, usually carries.

value: a positive integer indicating the amount of give fuel.

force_id: whether a platform belongs to the Red or Blue force.

value: Red or Blue.

fuel_cons_hover, **fuel_cons_intercept**, **fuel_cons_land**, **fuel_cons_launch**, **fuel_cons_move_to_point**, **fuel_cons_follow_path**, **fuel_cons_refuel**, **fuel_cons_su_land**, and **fuel_cons_su_launch**: the fuel consumption rates expressed in pounds per hour of platforms during different movement states.

value: equal to or greater than zero.

fuel_max: the maximum amount of fuel in pounds that an air platform can carry.

value: equal to or greater than zero.

fuel_types: tanking platforms that can refuel a particular platform. If it is a tanker, then this lists the aircraft that it can refuel.

value: a list of platforms separated by spaces, e.g.:

/KA-6D/fuel_types "A-6 A-7 F-14 FA-18 A-4 F-4"

hit_tolerance_level: the amount of damage a platform can withstand before being destroyed. This parameter is used in conjunction with a weapon's *damage_pts* to model platform damage (see *Weapon Parameters* section).

value: equal to or greater than one. Air platforms are generally assigned a tolerance level of one; surface and subsurface platforms are assigned greater tolerance levels depending on their displacement.

id_number: an unique number assigned to a platform that BATMAN & ROBIN use to identify it.

value: refer to Table 3 for the range of valid Id numbers of different platforms.

Table 3.

Object Identification Numbers

Object Type	Id Number
Blue Air	0 - 199
Blue Surface	200 - 399
Blue Subsurface	400 - 599
Blue Weapons	600 - 799
Blue Radars	800 - 999
Blue Sonars	1000 - 1199
Red Air	2000 - 2199
Red Surface	2200 - 2399
Red Subsurface	2400 - 2599
Red Weapons	2600 - 2799
Red Radars	2800 - 2999
Red Sonars	3000 - 3199

initial_state: the initial movement state of a platform.

value: *su_launch* is appropriate for Blue platforms, and *launch* is appropriate for Red platforms.

large_picture: the large icon of the platform.

value: the name of the file that contains the graphic object.

launch_range: an AS-missile's launch point or weapons release line from its target in miles. This parameter is only used with cruise missiles.

value: a positive integer.

long_name: the NATO name for a platform, e.g., "BACKFIRE" for the Soviet TU-26.

value: depends on the platform.

low_fuel: the amount of remaining or low fuel at which an aircraft calls for tanking or refueling.

value: a positive integer.

medium_picture: the medium-size icon of the platform.

value: the name of the file that contains the graphic object.

min_visible_cross_section: a platform's cross-section used by the detection algorithm to mimic sighting it.

value: a positive integer.

mother_types: the *mother* or *home-base* platforms that this platform can launch from or recover to.

value: a list of platforms separated by spaces, e.g.:

/F-14/mother_types "Nimitz Base"

prefix: the label displayed below a platform's icon during BATMAN or ROBIN.

value: any string the user desires with three or less characters.

radar_cross_section: a platform's radar-cross section used by the detection algorithm.

value: a positive integer.

sensors: the sensors associated with a platform.

value: a list of sensors separated by spaces where each is a *sensor* object whose parameters are (see *Sensor Parameters* section) defined somewhere else in the Object-Definition Database. The following example allocates two sensors to each F/A-18, whose attributes are expected to be defined later in the Object-Definition Database:

```
/FA-18/sensors              "FA-18air_radar FA-18surf_radar"
.
.
.
/FA-18air_radar/range       "100"
/FA-18air_radar/points      "4"
/FA-18air_radar/azimuth     "90"
etc.
```

side_picture: a side-view icon of a platform.

value: the name of the file that contains this graphic object.

small_picture: a small icon of a platform.

value: the name of the file that contains the graphic object.

speed_def, speed_max and **speed_min**: the default, upper, and lower velocity bounds of a platform in nautical miles per hour.

value: positive integers.

speed_follow_path, speed_hover, speed_intercept, speed_land, speed_launch, speed_move_to_point, speed_refuel, speed_su_land, and **speed_su_launch**: the speed in nautical miles per hour of a platform during each simulated state (DMI-SimMod).

value: a positive integer.

stats_type: a platform's type used by the performance-measurement utility.

value: AIR, SURF, or SUB only

update_follow_path, update_hover, update_intercept, update_land, update_launch, update_move_to_point, update_refuel, update_su_land, and **update_su_launch**: the names of the C functions that are used to update platforms at each simulated state. **CAUTION: Changing any of these parameters may cause BATMAN to behave erratically or even crash. It is recommended that these parameters only be modified by the maintenance programmer.**

value: C function names.

visual_range: the approximate distance a platform's pilot or captain can see in nautical miles.

value: an integer equal to or greater than zero.

weapons: the types and number of weapons that a platform can carry.

value: a list of weapons separated by spaces where each type is prefixed by a positive integer. If the weapon is not prefixed by an integer, then the number defaults to one. For example, the following allocates two Harpoon missiles and one Rockeye missile to the A-6 aircraft:

/A-6/weapons "2 harpoon rockeye"

12.2.2 Weapon Parameters

auto_loadout: whether a weapon should be automatically loaded out to a platform by the system at initialization time, rather than manually by the person using BATMAN.

value: y (yes) or n (no); the default is n.

class: specifies if the object is a platform or weapon.

value: weapon.

damage_pts: the amount of damage a weapon will inflict on its target at impact. This parameter is used in conjunction with a platform's *hit_tolerance_level* to model platform damage (see *Platform Parameters* section).

value: an integer greater than or equal to zero.

id_number: a number assigned to a weapon that BATMAN & ROBIN uses to identify it.

value: refer to Table 3 in the *Platform Parameters* section for the range of valid identification numbers of different weapons.

large_picture: the large icon of the weapon.

value: the name of the file that contains the graphic object.

medium_picture: the medium-size icon of the weapon.

value: the name of the file that contains the graphic object.

min_target_cross_section: the minimum size platform or target that a weapon can destroy.

value: a positive integer.

mnemon: the mnemonic name of a weapon which is used to identify it in BATMAN.

value: depends on the weapon.

prob_kill: the likelihood that the weapon will destroy its target.

value: an integer representing a percentage between 0 and 100.

range: the maximum effective distance in nautical miles of a weapon.

value: a positive integer.

rounds_per_burst: the number of rounds fired in one burst by the simulated M-61 gun on F-14 and F/A-18 fighters.

value: a positive integer.

12.2.3 Sensor Parameters

altitude_max and **altitude_min**: a sensor's upper and lower bounds in feet above or below sea level.

value: positive and negative (subsurface) integers.

azimuth: the azimuth of radar or sonar coverage in degrees.

value: a positive integer between 1 and 360.

detection_func: the name of the C function called to detect platforms.

value: a C function name.

id_number: a number assigned to a sensor that BATMAN & ROBIN uses to identify it.

value: refer to Table 3 in the *Platform Parameters* section for the range of valid identification numbers of different sensors.

min_target_cross_section: the minimum cross-section size of a platform that a particular sensor can detect.

value: a positive integer.

points: the number of points in the polygon that depicts radar or sonar coverage.

value: an integer between 4 and 16.

range: a sensor's radar range in nautical miles.

value: a positive integer.

weapons_release_func: the name of the C function called to fire weapons at targeted platforms.

value: a C function name.

12.2.4 System-Configuration Parameters

air_pic: the NTDS (Navy Tactical Data System) icons for Blue and Red air appearing in the right strip in BATMAN used to view, or not view, the air battle.

value: the name of the file that contains the icon.

air_radar_pic: the Blue and Red NTDS air icons appearing in the right strip in BATMAN used to turn air radar coverage on or off.

value: the name of the file that contains the icon.

air_status_pic: the NTDS icon for Blue air appearing in the right strip in BATMAN used to view the status of aircraft for TFA, TFB, or TFC.

value: the name of the file that contains the icon.

alert_pic: the icon used to access the placement of aircraft on Alert or Ready 5, 15, and 30.

value: the name of the file that contains the icon.

batman_intro_pic: the header or preceding display shown before BATMAN.

value: the name of the file that contains the graphic object.

big_font: the large font used in BATMAN & ROBIN, e.g., in task force status windows or boards.

value: the name of the file that contains the font.

big_us_flag: the American-flag icon used in the select "Performance Measures" interface.

value: the name of the file that contains the icon.

big_us_ussr_flag: the American/Soviet-flags icon used in the select "Performance Measures" interface.

value: the name of the file that contains the icon.

big_ussr_flag: the Soviet-flag icon used in the select "Performance Measures" interface.

value: the name of the file that contains the icon.

cap_pic: the CAP(Combat Air Patrol)-station icon appearing in the right strip during the deployment phase of BATMAN.

value: the name of the file that contains the icon.

chain_pic: the chainsaw icon appearing in the right strip during the deployment phase of BATMAN.

value: the name of the file that contains the icon.

checks_pic: reserved for future use.

clear_pic: the icon that depicts the screen-clear operation during deployment in BATMAN.

value: the name of the file that contains the icon.

colormap: the color map used in BATMAN & ROBIN which is viewed as a palette of colors each represented by three numbers that specify the red, green, and blue hues.

value: an absolute pathname to the file containing the color map.

cr_cr_pic: reserved for future use.

data_path: the complete pathname to the directory that holds BATMAN & ROBIN scenarios.

value: an absolute pathname to a directory.

debug_on: reserved for future use.

demo_scenario_number: the number of the scenario that is used currently in the BATMAN & ROBIN demonstration.

value: the identification number of an existing scenario.

disclaimer_pic: reserved for future use.

disp_grid_in_batman: whether or not the vector-logic or defensive grid should be displayed in BATMAN.

value: y or n.

disp_lat_lons_ir_batman: whether or not latitude and longitude lines should be displayed in BATMAN.

value: y or n.

display_height: the vertical resolution of the display window in pixels.

value: a number between and including zero and 1152; reasonable values are 1132 and 1152.

display_left: the x location of the upper-left corner of the main or parent display window for BATMAN & ROBIN specified as a pixel offset from (0,0).

value: a number between and including zero and 1152; reasonable values are zero and ten.

display_top: the y location of the upper-left corner of the main or display window for BATMAN & ROBIN specified as a pixel offset from (0,0).

value: a number between and including zero and 900; reasonable values are zero and ten.

display_width: the horizontal resolution of the display window in pixels.

value: a number between and including zero and 900; reasonable values are 880 and 900.

fonts: the complete pathname to the directory that contains the fonts used in BATMAN & ROBIN.

value: an absolute pathname to a directory.

grey25_pic: an icon that provides highlighting in ROBIN. When the user views warfare theaters by selecting areas from the miniature world map in the upper-left corner of ROBIN, the grey25_pic is used to highlight the user's current selection.

value: the name of the file that contains the icon.

grey50_pic: reserved for future use.

grey75_pic: reserved for future use.

grid_max_radius: the maximum radius of the vecor-logic grid specified by the user.

value: a positive integer larger than grid_min_radius.

grid_min_radius: the minimum radius of the vecor-logic grid specified by the user.

value: a positive integer smaller than grid_max_radius.

grid_pic: the small icon used to access the specification of the vector-logic grid radius.

value: the name of the file that contains the icon.

hammer_pic: the Soviet hammer-and-sickle icon used to bring in the Red force in BATMAN.

value: the name of the file that contains the icon.

heap_debug_level: a debugging tool used to flag memory allocation errors. It is intended to be used by the software-maintenance engineer, and requires an understanding of the C "malloc" function (Kernighan & Ritchie, 1978) and UNIX's "dbx" source-code debugger (Sun UNIX Commands Reference Manual, 1988).

value: set this value to zero under normal operation, to one to evaluate pointers to malloced memory, or to two to evaluate all malloced memory and detect memory overwrites.

hook_pic: an icon of the small box used to enclose the number of Red aircraft in a swarm at a particular point in ROBIN.

value: the name of the file that contains the icon.

hori_explo_pic: the icon that depicts the destruction of air platforms in BATMAN. (vert_explo_pic is used for surface and subsurface platforms.)

value: the name of the file that contains the icon.

horizontal_line: an icon of a horizontal line which can be clipped to form part of pop-up-message windows displayed in BATMAN & ROBIN.

value: the name of the file that contains the icon.

horizontal_shadow: an icon of a horizontal drop-shadow line used to form part of the pop-up-message windows displayed in BATMAN & ROBIN.

value: the name of the file that contains the icon.

host_passwd: reserved for future use.

in_color: whether BATMAN & ROBIN are to be run in color or black-and-white.

value: y for color; n for black-and-white.

itroff_command: the UNIX command used to print performance measures on a laser printer.

value: a valid UNIX print command, e.g., "psroff".

loadout_map_pic: the icon used to return to the warfare theater from loadout in BATMAN.

value: the name of the file that contains the icon.

loadout_next_pair_pic: the icon used to display the next pair of aircraft on a carrier or other *home base* to be loaded out in BATMAN.

value: the name of the file that contains the icon.

loadout_pic: reserved for future use.

map_font: the font used to write latitude and longitude labels, e.g., 40.00n.

value: the name of the file that contains the font.

master_passwd_file: the file that contains the system-administrator's password.

value: an absolute pathname to the file.

med_font: the medium-size font used in BATMAN & ROBIN.

value: the name of the file that contains the font.

missile_launch_pic: an icon that depicts when a Red aircraft is at its weapons-release line in ROBIN.

value: the name of the file that contains the icon.

mother_names: a list of the *mother* or *home-base* platforms in the BATMAN & ROBIN system.

value: a list of platforms separated by spaces, e.g.:

/sysparm/mother_names "Nimitz Tarawa Base"

move_pic: the icon that represents moving or changing positions of CAPs and Chainsaws in BATMAN.

value: the name of the file that contains the icon.

msec_engine_update_interval: the amount of processor time in milliseconds dedicated to each call of *simulation_engine* before it returns control to the SunView windowing and input system (refer to **Engine** subsection in Part II of this document).

value: the recommended value is 15.

msec_timer_step_size: the amount of time in milliseconds that BATMAN will run for each time-warped unit of a scenario specified in ROBIN. This parameter is only used when the simulation is not set to operate in real-time.

value: the recommended value is 1000.

nprdc_logo_pic: the NPRDC logo icon.

value: the name of the file that contains the icon.

ntds_blue_air_symbol: the NTDS icon for Blue- or Console-force aircraft.

value: the name of the file that contains the icon.

ntds_blue_sub_symbol: the NTDS icon for Blue submarines.

value: the name of the file that contains the icon.

ntds_blue_surf_symbol: the NTDS icon for Blue surface platforms.

value: the name of the file that contains the icon.

ntds_red_air_symbol: the NTDS icon for Red- or path-force aircraft.

value: the name of the file that contains the icon.

ntds_red_sub_symbol: the NTDS icon for Red submarines.

value: the name of the file that contains the icon.

ntds_red_surf_symbol: the NTDS icon for Red surface platforms.

value: the name of the file that contains the icon.

pic_path: the complete pathname to the directory that contains BATMAN & ROBIN icons and graphic objects.

value: an absolute pathname to a directory.

playback_record_on: reserved for future use.

playback_play_on: reserved for future use.

playback_scenario_num: reserved for future use.

players_directory: the complete pathname to the directory that contains all of the user files.

value: an absolute pathname to a directory.

printer: the UNIX device name of the printer where performance measures are sent for hard copy.

value: a valid UNIX device name for a printer, e.g., ps328bu.

radar_density_on: whether or not radar coverage will appear as shaded polygons. When set to "y", this parameter will override radar_poly_on. See below.

value: y or n; if this value is n, then radar_poly_on must be set to y.

radar_poly_on: whether or not radar coverage will appear as solid polygons. This parameter is only effective when radar_density_on is set to n. See above.

value: y or n; if this value is n, then radar_density_on must be set to y.

real_time_based: whether or not BATMAN should be run in real-time, i.e., based on a 24-hour clock.

value: when y, BATMAN is run in real-time; when n, BATMAN's timing is controlled by msec_timer_step_size. See above.

red_bombers: the NATO names of the Red bomber aircraft in the Panel used to create the raids in ROBIN.

value: a list of Red bomber NATO names separated by spaces, e.g.:

/sysparm/red_bombers "Backfire Badger Bear Blackjack Blinder"

red_fighters: the NATO names of the Red fighter aircraft in the Panel used to create the raids in ROBIN.

value: a list of Red fighter NATO names separated by spaces, e.g.:

/sysparm/red_fighters "MiG-17 MiG-19 MiG-21 MiG-23 MiG-25"

red_ships: the NATO names of the Red surface platforms in the Panel used to create raids in ROBIN.

value: a list of Red surface NATO names separated by spaces, e.g.:

/sysparm/red_ships "Krivak Kashin Udaloy Sovremeny"

red_subs: the NATO names of the Red subsurface platforms in the Panel used to create raids in ROBIN.

value: a list of Red subsurface NATO names separated by spaces, e.g.:

/sysparm/red_subs "Charlie Delta-IV Echo Foxtrot Victor"

remove_pic: the erase-specific-platform icon used during BATMAN.

value: the name of the file that contains the icon.

robin_intro_pic: the introductory title to ROBIN.

value: the name of the file that contains the graphic object.

sm_blue_air_ntds: a small Blue-air NTDS icon which can appear in the upper left corner when performance measures are displayed.

value: the name of the file that contains the icon.

sm_blue_s_ss_ntds: a small Blue-surface/subsurface NTDS icon which can appear in the upper left corner when performance measures are displayed.

value: the name of the file that contains the icon.

sm_font: the small font used in BATMAN & ROBIN.

value: the name of the file that contains the font.

sm_red_air_ntds: a small Red-air NTDS icon which can appear in the upper left corner when performance measures are displayed.

value: the name of the file that contains the icon.

sm_red_s_ss_ntds: a small Red-surface/subsurface NTDS icon which can appear in the upper left corner when performance measures are displayed.

value: the name of the file that contains the icon.

sm_world_pic: the miniature world map used to select warfare theaters in ROBIN.

value: the name of the file that contains the graphic object.

small_cap_pic: the icon displayed where the user positions CAP-stations during BATMAN.

value: the name of the file that contains the icon.

small_chain_dot_pic: an icon of a small dot representing the end-points of a chainsaw during BATMAN.

value: the name of the file that contains this icon.

sonar_pic: one icon appearing in the right strip in BATMAN containing Blue air, surface, and subsurface and Red subsurface NTDS symbols used to display sonar coverage.

value: the name of the file that contains the icon.

stats_title_font: the font used for the "Performance Measures" title.

value: the name of the file that contains the font.

status_pic: the icon used in BATMAN to display the status of a task force's air or surface platforms.

value: the name of the file that contains the icon.

string_pic: reserved for future use.

subsurf_pic: the Blue and Red NTDS subsurface icons appearing in the right strip in BATMAN used to view, or not view, the subsurface battle.

value: the name of the file that contains the icon.

subsurf_status_pic: the NTDS icon for Blue sub-surface platforms appearing in the right strip in BATMAN used to view the status of sub-surface platforms for TFA, TFB, or TFC.

value: the name of the file that contains the icon.

surf_pic: the Blue and Red NTDS surface icons appearing in the right strip in BATMAN used to view, or not view, the surface battle.

value: the name of the file that contains the icon.

surf_radar_pic: the Blue and Red NTDS surface icons appearing in the right strip in BATMAN used to turn surface radar coverage on or off.

value: the name of the file that contains the icon.

surf_status_pic: the NTDS icon for Blue surface platforms appearing in the right strip in BATMAN used to view the status of surface platforms for TFA, TFB, or TFC.

value: the name of the file that contains the icon.

task_force_pic: a generic task force icon appearing in BATMAN during aircraft loadout used to view again ships in a task force.

value: the name of the file that contains the icon.

tfa_status_pic, **tfb_status_pic**, and **tfc_status_pic**: the icons appearing in the right strip in BATMAN used to specify which task force's status to display.

value: the names of the files that contain the icons.

time_to_stop: the amount of simulated or warped time in minutes that BATMAN will run before stopping.

value: a positive integer.

vert_explo_pic: the icon that depicts the destruction of surface and subsurface platforms in BATMAN. hori_explo_pic is used for air platforms.

value: the name of the file that contains the icon.

vertical_line: an icon of a vertical line which can be clipped to form part of pop-up-message windows displayed in BATMAN & ROBIN.

value: the name of the file that contains the icon.

vertical_shadow: an icon of a vertical drop-shadow line used to form part of the pop-up-message windows displayed in BATMAN & ROBIN.

value: the name of the file that contains the icon.

vl_pic: the Victor-Lima (V/L) icon used to indicate the defended point.

value: the name of the file that contains the icon.

warn_message_font: the font used for the advisory or warning messages that are displayed prior to BATMAN or ROBIN.

value: the name of the file that contains the font.

zoom_pic: the icon in BATMAN that represents the zoom function.

value: the name of the file that contains the icon.

12.2.5 Performance-Measures Parameters

heads_up_dist: the distance in nautical miles a Red-force platform must be less than to be within "heads-up" range of a task force.

value: a positive integer.

stats_save_events: whether or not the consequences of users' tactical actions during BATMAN should be saved to compute performance measures.

value: y if the events should be saved; n if they should not be.

stats_save_results: whether or not the performance measures themselves should be saved in the users' directories.

value: y if performance measures should be saved; n if they should not be.

12.2.6 User-Database Parameters

button_color: the color of panel-buttons to add/delete users.

value: the form string for this parameter should be "color offset". Color is specified in the "color map" file mentioned above; offset indicates a variation in color, e.g., "red 8" would set button_color to the eighth variation of red.

player_panel_border_pic: an icon of a thin dark horizontal line used as a border between the "operations" and "list of users" panels displayed when the user is adding, deleting, or selecting users.

value: the name of the file that contains the icon.

uinfo_border_color: the background color of the screen when entering a new user's name and social security number.

value: a string of the form: "color offset". See button_color above.

user_bg_color: the color of empty slots in the "list of users" panel.

value: a string of the form: "color offset". See button_color above.

user_fg_color: the color of users' names in the "list of users" panel.

value: a string of the form: "color offset". See button_color above.

username_border_color: the border color around users' names in the "list of users" panel.

value: a string of the form: "color offset". See button_color above.

12.3 Sample Platform-Configuration Parameters

The following example illustrates an F-14 fighter entry from the Object-Definition database which appears in the *.defaults* file. For further elucidation it is recommended that software maintenance engineers refer to a copy of the *.defaults* file.

/object/parameter	**"value"**
/f14/force_id	"BLUE"
/f14/stats_type	"AIR"
etc.	

13.0 Graphic Database

This database contains all the icons or graphic objects used in BATMAN & ROBIN. These are in Sun raster-file format, and each exists in its own file (Pixrect Reference Manual, 1988).

14.0 Scenario Database

The scenario database is comprised of standard text files which are created by ROBIN and used by BATMAN. The UNIX utilities "cat" and "more" can be used to view these files. This database is stored in the directory specified by the data_path parameter from the Object-Definition Database.

Each scenario is split into two files: a Blue-force file and a Red-force file. These contain all the necessary data for the Blue and Red forces for a specific scenario. The format for filenames is "red" or "blue" followed by a scenario number, e.g., the files for scenario 10 would be **red.10** and **blue.10**. Scenario numbers range from 1-999.

In addition to these files, there are three support files used for scenario management: **blue.1000**, **red.1000**, and **scenario_index**. These files are also located in the directory specified by data_path. The **blue.1000** and **red.1000** files are used as templates to build the **blue.n** and **red.n** files. The **scenario_index** file lists the defined scenarios by number.

14.1 Blue-Force File

The Blue-force file lists air, surface, and subsurface platforms as well as weapons that are available for allocation, deployment, and management in a BATMAN scenario. This file is read in by BATMAN to initialize scenario data structures.

The first field in this file is the task-force designator. BATMAN & ROBIN can handle a maximum of three Blue task forces: TFA, TFB, and TFC. Following the task-force

designator line, is a list of the platforms in a specific task force. In each line, the number of platforms is listed followed by their NATO names.

A few special rules apply when a platform is an aircraft carrier. First, there can be no more than one carrier per task force. Second, the carrier platform line must contain a list of the aircraft and weapons that are on-board. The format of the aircraft and weapons line must be the same as the other platform lines. Finally, if there is a carrier in the task force, it must be the first platform defined.

Following the carrier definition is a list of escort platforms in the task force. After all the platforms for a task force are defined, the next task-force definition starts, if there is more than one task force in a scenario. The following is an example of a Blue-force file with one task force, i.e., TFA.

```
{
FORCE A
{
        1 cvn
            {
            24 f14
            18 fa18
            9 e2c
            6 ka6
            9 a6
            5 a7
            5 s3b
            400 phoenix
            50 sparrow
            300 sidewinder
            50 harm
            0 harpoon
            100 walleye
            50 rockeye
```

```
        50 mk82
        }
    2 cgn
    3 cg
    4 dd
    2 ddg
    5 ff
    1 ffg
    9 ssn
    }
}
```

14.2 Red-Force File

The Red-force file contains a summary of the tactical situation, a list of the available Red platforms, and track-movement definitions for each Red platform.

14.2.1 Tactical-Situation Section

In a manner similar to the Object-Definition Database, a tactical situation is described by a set of parameters, each holding a particular value. This provides a uniform method for describing a variety of battle scenarios. The following is a brief description of each parameter from the Tactical-Situation section of a Red-force file.

DEFCON: reserved for future use.

Headlines: reserved for future use.

Intelligence_Information: reserved for future use.

Messages: reserved for future use.

Red_Platforms: indicates the name and amount of each Red platform defined for the scenario.

> ***value***: a list of Red platforms separated by spaces, where each platform is followed by a positive integer which indicates how many of a specific type. The following scenario would have two Victor submarines and five Blackjack bombers:
>
> Red_Platforms: Victor 2 Blackjack 5

ROEs: reserved for future use.

Sea_State: reserved for future use.

Threat_Axes_and_Bases: reserved for future use.

Time_Warp: specifies a time-warp for the scenario.

> ***value***: a positive integer between and including one and 100.

Title: reserved for future use.

Vl_lat and **Vl_lon**: the latitude and longitude of V/L.

> ***value***: real numbers, e.g., (Vl_lat = 15.00, Vl_lon = -80.00). The latitude and longitude of V/L will lie somewhere in the scenario's warfare theater.

Warfare_Theater: the warfare area for the battle.

> ***value***: at this time the following theaters are available, but not limited to: arabian_sea, bering_sea, caribbean, japan_sea, kamchatka_peninsula, mediterranean, murmansk, north_atlantic, persian_gulf, and south_east_asia.

Weather: reserved for future use.

14.2.2 Red-Tracks Section

This section contains a series of records defining the movement tracks for Red-force platforms in a scenario. Unlike the tactical situation-display parameters, the Red-track definitions are listed on contiguous lines without breaks or comments. Hence, it is difficult for the user to decipher the contents of these records. This track data is primarily designed to be read by BATMAN when the scenario is initialized, or by ROBIN when the scenario is modified.

Platform-track data is represented by a tree structure with nodes and vectors. A node is a location where one or more platforms of the same type move together. Each node may have one or more vectors depicting segments of the movement path for an individual platform or swarms of them. The last vector for a node is either the terminal point, or the position where the swarm splits into subswarms or individual platforms.

14.2.3 Sample Red-Force file

The following is an example of tactical-situation section of a Red-force file:

Title	: reserved for future use
Warfare_Theater	: caribbean
Vl_lat	: 15.00
Vl_lon	: -80.00

Threat_Axes_and_Bases	: reserved for future use
DEFCON	: reserved for future use
ROEs	: reserved for future use
Intelligence_Information	: reserved for future use
Weather	: reserved for future use
Sea_State	: reserved for future use
Messages	: reserved for future use
Headlines	: reserved for future use
Time_Warp	: 10
Red_Platforms	: Victor 2 Blackjack 5 Charlie 10

swarm_track : beg_plat 7 end_plat 17 name Charlie start_time 0

track_vector: lat 16.23 lon -74.53 altitude -100 speed 20

track_vector: lat 15.37 lon -75.06 altitude -100 speed 20

track_vector: lat 14.90 lon -75.89 altitude -100 speed 20

track_vector: lat 14.97 lon -76.39 altitude -100 speed 20

swarm_track : beg_plat 2 end_plat 7 name Blackjack start_time 0

track_vector: lat 16.23 lon -74.53 altitude 20000 speed 400 chaff 1

track_vector: lat 15.37 lon -75.06 altitude 20000 speed 400 jam 62

track_vector: lat 14.90 lon -75.89 altitude 20000 speed 400 chaff 1 jam 15

track_vector: lat 14.97 lon -76.39 altitude 20000 speed 400

swarm_track : beg_plat 0 end_plat 2 name Victor start_time 0

track_vector: lat 14.97 lon -76.39 altitude -100 speed 20

track_vector: lat 15.21 lon -79.27 altitude -100 speed 20

15.0 User-Performance Database

The User-Performance database consists of multivariate performance measures stored in

standard text files. Each BATMAN & ROBIN user is given their own UNIX directory which contains identifying information and all performance measures for a user. BATMAN & ROBIN can maintain more than names and social security numbers of users, e.g., number of flight hours or other descriptive data.

References

Bloom, B. S., Hastings, J. T., & Madaus, G. F. (1971). *Handbook on formative and summative evaluation of student learning* New York: McGraw-Hill.

Federico, P-A. (1989). *Computer-based and paper-based measurement of recognition performance* (NPRDC TR 89-7). San Diego CA: Navy Personnel Research and Development Center.

Federico, P-A., & Liggett, N. L. (1989). *Computer-based and paper-based measurement of semantic knowledge* (NPRDC TR 89-4). San Diego CA: Navy Personnel Research and Development Center.

Federico, P-A., Bickel, S. H., Ullrich, R. R., & Bridges, T. E.(in preparation). *BATMAN & ROBIN: Human-computer interface and simulation models* (NPRDC TN 89-XX). San Diego CA: Navy Personnel Research and Development Center.

Hutchins, E. L., Hollan, J. D., & Norman, D. A. (1986). Direct-manipulation interfaces. In D. A. Norman & S. W. Draper (Eds.), *User centered system design.* Hillsdale NJ: Lawrence Erlbaum Associates.

Kernighan, B.V. & Ritchie, D.M. (1978). *The C programming language.* Englewood Cliffs NJ: Prentice-Hall.

Liggett, N. L., & Federico, P-A. (1986). *Computer-based system for assessing semantic knowledge: Enhancements* (NPRDC TN 87-4). San Diego CA: Navy Personnel Research and Development Center.

Little, G. A., Maffly, D. H., Miller, C.L., Setter, D. A., & Federico, P-A. (1985). *A computer-based gaming system for assessing recognition performance (recog)* (TL 85-3). San Diego CA: Training Laboratory, Navy Personnel Research and Development Center.

McGilton, H., & Morgan, R. (1983). *Introducing the UNIX system.* New York: McGraw-Hill.

Pixrect Reference manual. (1988). Mountain View CA: Sun Microsystems, Inc.

Shneiderman, B. (1982). The future of interactive systems and the emergence of direct manipulation. *Behavior and information technology*, 1, 237-256.

Stroutstrup, B. (1988, May). What is object-oriented programming? *IEEE Software*, 10-20.

SunOS reference manual. (1988). Mountain View CA: Sun Microsystems, Inc.

SunView Programmer's guide. (1988). Mountain View CA: Sun Microsystems, Inc.

SunView System programmer's guide. (1988). Mountain View CA: Sun Microsystems, Inc.

Wilson, R. (1988, November). Object-oriented languages reorient programming techniques. *Computer Design*, 52-62.

DISTRIBUTION LIST

Distribution:
OP-01B2
OP-11B1
OP-11H
SPAWAR 159 (JTIDS)
SPAWAR 159-4 (JTIDS)
NPS (Code 55NA)
NPS (Code 55HA)
NPS (Code 55PD)
NADC (Code 40L)
NOSC (Code 4504)
NOSC (Code 432)
APL, WAL
DTIC (2)

Copy to:
OSD (C31) (T&T)
JCS (J-7) (JE&T)
NSWC (Code N-31)
NWC (Code 33T)
CNA