NPSNET: OBJECT ANIMATION
SCRIPT INTERPRETATION SYSTEM

by

Phillip D. West

September 1991

Thesis Advisors: Michael J. Zyda
David R. Pratt

Approved for public release; distribution is unlimited.
The goal of this work is to develop a text-based script interpretation system for easy and efficient 3D visual simulations without extensive programming. Scripts are sequences of events representing task-level behaviors in virtual worlds systems. The Object Animation Script Interpretation System for NPSNET (NPSNET-OASIS), provides animators at the Naval Postgraduate School a mechanism for interacting with 3D visual simulations via scripted autonomous players. Libraries of scripts are collected for rapid generation of 3D visual simulations. NPSNET-OASIS makes use of object-oriented design methodologies for reusability and extensibility. Included in NPSNET-OASIS are the object tools for script processing, writing, and sorting.
NPSNET: OBJECT ANIMATION SCRIPT INTERPRETATION SYSTEM

by

Phillip D. West
Lieutenant, United States Navy
B.S., Penn State University, 1984

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1991

Author: 

Phillip D. West

Approved By: 

Michael J. Zyda, Thesis Advisor

David R. Pratt, Co-Advisor

Robert B. McGhee, Chairman,
Department of Computer Science
ABSTRACT

The goal of this work is to develop a text-based script interpretation system for easy and efficient 3D visual simulations without extensive programming. Scripts are sequences of events representing task-level behaviors in virtual worlds systems. The Object Animation Script Interpretation System for NPSNET (NPSNET-OASIS), provides animators at the Naval Postgraduate School a mechanism for interacting with 3D visual simulations via scripted autonomous players. Libraries of scripts are collected for rapid generation of 3D visual simulations. NPSNET-OASIS makes use of object-oriented design methodologies for reusability and extensibility. Included in NPSNET-OASIS are the object tools for script processing, writing, and sorting.
# TABLE OF CONTENTS

I. **THE NEED FOR SCRIPT ANIMATION LANGUAGES** ............................................ 1  
   A. **INTRODUCTION** ................................................................................................. 1  
   B. **BACKGROUND** .................................................................................................... 2  
   C. **SUMMARY OF CHAPTERS** .................................................................................. 4  

II. **DEVELOPMENT OF NPSNET-OASIS** .............................................................. 5  
   A. **DESIGN CRITERIA** ............................................................................................. 5  
   B. **OBJECT-ORIENTED DESIGN** .............................................................................. 5  
      1. Class Hierarchy ........................................................................................................ 5  
      2. Event Objects .......................................................................................................... 6  
      3. System Objects ....................................................................................................... 7  
   C. **NPSNET-OASIS SCRIPT FORMAT** ..................................................................... 9  

III. **OVERVIEW OF NPSNET-OASIS** ..................................................................... 11  
    A. **SCRIPTING TOOLS** .......................................................................................... 11  
       1. Script Processor .................................................................................................... 12  
       2. Script Generator .................................................................................................. 13  
       3. System Timekeeper ............................................................................................... 13  
    B. **STRUCTURE OF SCRIPT EVENTS** ................................................................. 15  
       1. 3D Icon Identification .......................................................................................... 15  
       2. Event Position ...................................................................................................... 16  
       3. Timestamp ............................................................................................................ 17  
       4. Event Attributes .................................................................................................. 17  
    C. **SCRIPT OPTIONS** ............................................................................................ 18  
       1. Script_abort ......................................................................................................... 18  
       2. Script_Call .......................................................................................................... 21  
       3. Script_Chain ......................................................................................................... 21  
       4. Script_Repeat ....................................................................................................... 21
IV. IMPLEMENTATION OF NPSNET-OASIS ............................................. 24
   A. OVERVIEW ........................................................................... 24
   B. DATA STRUCTURES .................................................................. 24
   C. NPSNET-OASIS NETWORK INTERFACE ..................................... 24
      1. Internal Data Structures .......................................................... 25
      2. Message Packets ................................................................. 26
      3. Operation of Network Interface ............................................... 26

V. CONCLUSIONS AND RECOMMENDATIONS ............................................. 29
   A. SUMMARY AND CONCLUSION .................................................. 29
   B. LIMITATIONS ........................................................................ 29
   C. FUTURE DIRECTIONS .............................................................. 29

APPENDIX A: NPSNET-OASIS Script Events, Results, and Errors ................. 31
APPENDIX B: NPSNET-OASIS Sample Scripts ............................................. 49
APPENDIX C: Class Definition of ScriptProcessor ....................................... 53
APPENDIX D: Class Definition of ScriptGenerator ....................................... 55
APPENDIX E: Class Definition of TimeKeeper ........................................... 57
APPENDIX F: Class Definition of OasisSystem ........................................... 59
APPENDIX G: Class Definition of OasisScriptSorter ................................... 62
APPENDIX H: Class Definition of OasisScriptPreprocessor .......................... 63
| Figure 2.1 | Root Class of NPSNET-OASIS Class Hierarchy | 6 |
| Figure 2.2 | Ancestors of Class ScriptEvent | 6 |
| Figure 2.3 | Descendants of Class EventAttribute | 7 |
| Figure 2.4 | Ancestors of Class ScriptObject | 8 |
| Figure 2.5 | Ancestors and Descendants of Class OasisSystem | 8 |
| Figure 2.6 | Sample Script | 9 |
| Figure 2.7 | General Syntax for Script Statements | 10 |
| Figure 3.1 | The NPSNET-OASIS System | 11 |
| Figure 3.2 | Error Message | 12 |
| Figure 3.3 | Illustration of Timestamp Adjustment Based on System Clock | 14 |
| Figure 3.4 | Illustration of Timestamp Adjustment Based on User Clock | 14 |
| Figure 3.5 | The 3D Icon Identification Number | 15 |
| Figure 3.6 | UTM Coordinate Position | 16 |
| Figure 3.7 | UTM Coordinate Position With Padded Zeros | 17 |
| Figure 3.8 | Timestamp Formats | 18 |
| Figure 3.9 | Sample Script With Script Options | 19 |
| Figure 3.10 | Sample Script With Script Options | 20 |
| Figure 4.1 | NPSNET-OASIS Network Interface | 25 |
| Figure 4.2 | Network Interface Monitor Routine | 28 |
I. THE NEED FOR SCRIPT ANIMATION LANGUAGES

A. INTRODUCTION

The Graphics and Video Laboratory of the Computer Science Department at the Naval Postgraduate School (NPS) has a long history of developing 3D visual simulation systems on inexpensive, commercially available graphics workstations [Ref. 10]. The visual simulators developed in the Graphics and Video Laboratory include the FOG-M missile simulator, the VEH vehicle simulator, the Airborne Remotely Operated Device (AROD), the Moving Platform Simulator series (MPS-1, MPS-2, and MPS-3), the High Resolution Digital Terrain Model (HRDTM) system, the Forward Observer Simulator Trainer (FOST) the NPS Autonomous Underwater Vehicle simulator (NPSAUV), and the Command and Control Workstation of the Future system (CCWF). Current visual simulation efforts are focused on the NPSNET system, a 3D visual simulation system that utilizes SIMNET and DMA databases.

NPSNET is a real-time, 3D visual simulation system capable of displaying various types of vehicles - ground, ships, and aircraft [Ref. 10]. The system is capable of displaying additional objects such as missiles, buildings, trees, and environmental effects, such as fog and smoke. Objects are represented initially by pre-defined 3D icons stored in Object File Format (OFF) [Ref. 9]. 3D icons are geometric descriptions of 3D objects. Vehicle movements in NPSNET are controlled by mouse, spaceballs, and button/dialboxes. In addition, vehicles can be driven interactively from other workstations by means of message packets via Ethernet.

In any simulator, the backbone of the system is its internal data structures for modeling the state of the world [Ref. 10]. It is from the world state information that visual
displays are generated. Continuous and transient events are referred to as task-level behaviors in virtual world systems [Ref. 1]. Continuous events are dynamic changes in motion of 3D icons. Transient events are dynamic changes of appearances in virtual 3D icons, such as explosions and collisions.

When sufficient numbers of actual interactive players are not available, the Graphics and Video Laboratory requires two methods for generating autonomous players to populate the world - semi-automated forces and scripting [Ref. 10]. Currently in development, semi-automated forces provide intelligent behavioral models to autonomous players via parallel processing and the network. Scripting provides a programmable mechanism to add autonomous agents or to change task-level behaviors of 3D icons.

The script system, the Object Animation Script Interpretation System for NPSNET (NPSNET-OASIS), was developed to meet the requirements of scripted autonomous players. Designed using object-oriented methodologies, NPSNET-OASIS provides the capabilities to record and playback scripts of task-level behaviors. Scripts in NPSNET-OASIS are generated as sequences of events in uniform order based on timestamps. Unlike current systems which are coded in standard C, NPSNET-OASIS is programmed in C++, thus allowing reusability and extensibility. Several instances of NPSNET-OASIS can be integrated into NPSNET, allowing the simultaneous execution of multiple scripts.

B. BACKGROUND

There are two interaction paradigms in virtual world systems - guiding and programming [Ref. 1]. Guiding is interaction with objects from built-in procedural support and specially-designed graphics hardware. Programming is interaction with objects using special-purpose simulation software for algorithmic description and control.

The principles of software engineering are applied to natural script languages in an effort to provide a more flexible, extensible, and efficient interactive tool for visual
simulators [Ref. 5]. Libraries of scripts can be generated and reused, allowing fast prototyping of simulated engagements and tactics. Simple in design and use, scripts are basically procedures for controlling 3D icons. Combining scripts to create larger ones, supports modular scripting in a high-level of programming. In LISP-based systems, the rules of the script language are extensible so that new animation procedures and primitives can be added to the system. Easier to learn than complex languages, script languages can develop animation scripts faster than a functionally equivalent Ada or C program.

The basis for our research in script systems is on three earlier systems - ASAS, PDI, and MANUS. Based on the LISP language, Actor/Scriptor Animation System (ASAS), is a full programming language system for animation and graphics [Ref. 6]. ASAS supports independent program structures called actors, and includes a set of geometric objects and operators. Geometric objects include data types such as vectors, colors, polygons, solids, and lights. Actors are responsible for geometric objects in an animation sequence. Geometric operators are the tools the animator uses to shape, move, and orient geometric objects.

Influenced by ASAS, Pacific Data Images (PDI) developed a script system on top of the C programming language for creating animation in the entertainment field [Refs. 3, 5]. The PDI script system supports complex modeling, transformations, and motion. At each production stage, the script is updated to reflect production changes, and to incorporate new models and motion data from other parts of script system.

BOLIO, an integrated graphical simulation platform (IGSP), provides users tools to interact in simulation of task-level behaviors, and event-driven processes in virtual worlds [Refs. 1, 8]. A component of BOLIO, MANUS, provides the built-in language and processor for associating objects and processes in defining task-level behaviors. Programmers have access to primitive operations of kinematics and dynamics in a
modular function library. Complex scripts are used for testing and debugging various simulation modules, or for defining virtual environments

C. SUMMARY OF CHAPTERS

The development of NPSNET-OASIS involves understanding of required task-level behaviors of NPSNET and other 3D visual simulators of the Graphics and Video Laboratory. In Chapter II, the design of the script system is discussed. Chapter III discusses the overview of NPSNET-OASIS. The interaction of NPSNET-OASIS with NPSNET is discussed in Chapter IV. Limitations and future directions are the subjects of Chapter V. Appendices include syntax for script events and script options, listings of script results and script errors, sample scripts, and object class descriptions for NPSNET-OASIS.
II. DEVELOPMENT OF NPSNET-OASIS

A. DESIGN CRITERIA

The goal of NPSNET-OASIS is to build a scripting system that is reusable and extensible. In addition, NPSNET-OASIS must be simple for system integration. Previous 3D visual simulators in the Graphics and Video Laboratory were developed with traditional programming languages such as C, and are not easy to maintain. Whenever, a modification is made, the entire system is affected. The design of NPSNET-OASIS must be capable of being adapted easily as modifications are made to NPSNET.

B. OBJECT-ORIENTED DESIGN

The concept of object-oriented design (OOD) involves solving problems by identifying the real-world objects of the problem, and the processing required of those objects [Ref. 2]. For this reason, all components of NPSNET-OASIS are represented as objects. Classes in object-oriented design, distinguished in italics text, are templates for categories of objects, and provide the means for creating objects. Because objects serve as data abstractions, classes must include data structure definitions and the processing code for instances of those data structures.

1. Class Hierarchy

*OasisObject* is the root class of the NPSNET-OASIS class hierarchy, as other classes in the hierarchy are derived from it. The main descendants of *OasisObject* - *OasisSystemObject* and *OasisEventObject*, represent the main components of NPSNET-OASIS (Figure 2.1). *OasisSystemObject* represents the base class for all system components in script processing, script generating, and script sorting. *OasisEventObjects* represents the base class for all task-level behaviors in 3D icons. For illustrative purposes,
classes of NPSNET-OASIS are segmented from the total class hierarchy, and are represented as ellipses with arrows pointing to descendants.

2. Event Objects

Each ScriptEvent is composed of an EventObject, EventPosition, TimeStamp, and EventAttribute, which are descendants of OasisEventObjects (Figure 2.2). EventObjects provide identification of 3D icons in events, and EventPositions provide the locations of the events. For every ScriptEvent there is a TimeStamp, which provides the time mechanism to synchronize all events in uniform order. For additional information in supporting task-level behaviors of 3D icons, EventAttributes are used to represent the change of continuous and transient events (Figure 2.3).

Presented in Figure 2.4 are the six classes of script events - VehicleEvent, WeaponEvent, MiscObjectEvent, EnvironmentEvent, ScriptOption, ScriptComment, and ScriptOption. ScriptComment and ScriptOption do not represent script events. However,
script events are supported by them for descriptive narrations in script files and script synchronization. Communicating with script events, ScriptObjects are the interfaces for passing messages of task-level behaviors to and from 3D icons. ScriptObjects interact with virtual world systems for controlling and recording continuous and transient events.

3. System Objects

There are four parent classes for OasisSystem - ScriptFile, ScriptProcessor, ScriptGenerator, and TimeKeeper (Figure 2.5). Classes are derived from OasisSystemEvents for supporting system requirements of NPSNET-OASIS. ScriptFiles provide the mechanism for interacting script files with script systems. EventProcessor is the base class for the required event processors to the ScriptProcessor. ScriptProcessor is the tool for interpreting and processing of all script files. The script writer of NPSNET-OASIS, the ScriptGenerator, records all script objects to a script file. The last parent class, the TimeKeeper, provides the system times for processing and recording script files.
Derived from *OasisSystem*, are two additional object classes for NPSNET-OASIS (Figure 2.5). *OasisScriptPreprocessor*, is a preprocessor for validating script files prior to processing. Script errors are displayed for further script modification. The second object class, *OasisScriptSorter*, provides the sorting of script events based on time stamps. This tool ensures all script events are in uniform order.
C. NPSNET-OASIS SCRIPT FORMAT

All script files for NPSNET-OASIS are in text-based format allowing the user to use any standard text-editor for script editing and creation (Figure 2.6). It is easier to edit task-

/* Activate clouds with velocity 5.0 km/h, heading 270.0 west */
ENVIRONMENT_ACTIVATE 5010499 Cloud 270.0 5.0 34536783 1000.0 30.0

/* Activate two M-1 tanks */
VEHICLE_ACTIVATE 5010001 M1 090.0 40.0 345678 0.0 40.0
VEHICLE_ACTIVATE 5010002 M1 090.0 40.0 345678 0.0 45.0

/* Change headings of M-1 tanks */
VEHICLE_HEADING 5010001 135.0 344955 0.0 2:0.0
VEHICLE_HEADING 5010002 120.0 345958 0.0 2:5.0

/* M-1 tanks passing by a palm tree */
OBJECT_ACTIVATE 5010003 PalmTree 345700 0.0 4:0.0
OBJECT_DEACTIVATE 5010003 5:35.0

/* Change velocities of M-1 tanks */
VEHICLE_SPEED 5010001 5.0 344801 0.0 6:0.0
VEHICLE_SPEED 5010001 5.0 344803 0.0 7:25.0

/* M-1 tanks passing by a building */
OBJECT_ACTIVATE 5010004 Building 345700 0.0 8:10.0
OBJECT_DEACTIVATE 5010004 9:40.0

/* Deactivate M-1 tanks */
VEHICLE_DEACTIVATE 5010001 9:45.0
VEHICLE_DEACTIVATE 5010002 9:50.0

/* Deactivate clouds */
ENVIRONMENT_ACTIVATE 5010499 10:0.0

/* End Of Script */

Figure 2.6 Sample Script
level behaviors of 3D icons in ASCII text, than it is in binary format. Each script statement is required to be on a separate line. Since most text editors are capable of handling lines up to 132 columns, there is no reason for a single script statement to not fit all on one line. There is no limit on number of lines per script file.

All script event statements in NPSNET-OASIS contain a procedural operator and the required arguments for the procedural operator (Figure 2.7). Procedural operators are

```
OBJECT ACTIVATE 100 tree 79A-DN-123456 0.0 1:00.0
```

![Figure 2.7 General Syntax for Script Statements](image)

classified into five groups - vehicle, weapon, object, environment, and script. All NPSNET-OASIS procedural operators begin with the group name for rapid parsing and simplicity. A listing of all script events can be found in Appendix A.

In the interest of clarity and readability, blank lines and comments are permitted between statements. Comments are used to explain or describe the script event. Whenever the symbol ‘/*’ is encountered on a line, all characters from that point on, until the symbol ‘*/’ is reached, is considered to be a comment.

Case is not important in procedural operators and required arguments. All characters are converted to lower case for parsing and extraction. The only time case is important, is when characters in script file names are case sensitive for UNIX based input and output disk operations.
III. OVERVIEW OF NPSNET-OASIS

A. SCRIPTING TOOLS

The NPSNET-OASIS system communicates with three scripting tools - the script processor, the script generator, and the timekeeper (Figure 3.1). Each tool is independent of

![Diagram of NPSNET-OASIS System]

Figure 3.1 The NPSNET-OASIS System
the other. The integration of the scripting tools, provides animators the required mechanisms to record and playback scripts.

1. Script Processor

Hidden from the user, all functions of the script processor are accessed from the NPSNET-OASIS system. Functions include reading script objects and assigning script files to process. There is only one script file active in the script processor at any given instance. However, multiple processors are allowed. On end of file, the script processor can be assigned another script file from the NPSNET-OASIS system.

The script processor returns a script result after each read. Script results are based on valid script statements or errors reading the script file. File errors are treated as end of scripts or invalid script files. When an error occurs for a script statement, a message is displayed on the standard output device indicating type of error, followed by line number and name of current script file (Figure 3.2). Messages provide users a tool for debugging scripts. Description of script results and script errors are contained in Appendix A.

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Name of Script File</th>
<th>Type of Error</th>
</tr>
</thead>
</table>

**Figure 3.2 Error Message**

Script statements are read in line for line by the script processor. When blank lines are encountered, the script processor ignores them, and continues to read the script file. Beginning with the symbol ‘/*’ script comments are continuous script lines containing information about script events or script options. The script processor treats each line as a comment until the termination symbol ‘*/’ is reached. The user must be careful in using
script comments. A script comment with no termination symbol will include valid script events until end of file. Script lines containing script events or script options are valid only when both procedural operators and the required arguments are valid. The number of arguments for each procedural operator are fixed. Thus, incomplete or extra arguments will make the script line invalid. In addition, incomplete format of a required argument will preempt an error message by the script processor.

2. Script Generator

The script generator's primary purpose is to write script objects to an output file in NPSNET-OASiS script format. Error checking does not exist since script objects contain default values for all values not assigned in events. All script files generated from the script generator are valid files for script processing. When several script files exist, a library can be created by including several script files into one script. There are no specified maximum number of lines per script file. The only limitation is the space available on disk.

3. System Timekeeper

As timestamps are processed for each script event, the system timekeeper adjusts them for simulator interaction. Timestamps for each script event are assumed relative to the start time of the input script file. When absolute, timestamps are relative to the start time of the NPSNET-OASiS system. The timekeeper uses the system clock of the graphics workstations for all assignments of start times. Times received from system clocks are based on total seconds and total microseconds since January 1, 1970 [Ref. 7].

There are two types of clock references - system and user. The difference between the two, is that system clock reference is actual system time, and user clock reference is the time selected by the user in seconds and microseconds. When selected in user clock reference, the user has the option to change the start time for the NPSNET-OASiS system. When changed, the timekeeper adjusts the new start time with the system clock. In addition, the start times for the input script file and output script file are also changed to the same
start time as the NPSNET-OASIS system. When writing script files, timestamps are always reference to the start time of the output script file.

In Figures 3.3 and 3.4, timestamp adjustments are illustrated for system clock reference and user clock reference. The timestamp is converted to total seconds and total

<table>
<thead>
<tr>
<th>Timestamp of 10:01:05.5</th>
<th>System Start Time</th>
<th>Timestamp in Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Secs</td>
<td>Usecs</td>
<td>Total Secs</td>
</tr>
<tr>
<td>665</td>
<td>500000</td>
<td>683697600</td>
</tr>
<tr>
<td></td>
<td></td>
<td>683698266</td>
</tr>
</tbody>
</table>

Figure 3.3 Illustration of Timestamp Adjustment Based on System Clock

<table>
<thead>
<tr>
<th>Timestamp of 10:01:05.5</th>
<th>System Start Time</th>
<th>System Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Secs</td>
<td>Usecs</td>
<td>Total Secs</td>
</tr>
<tr>
<td>665</td>
<td>500000</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>665</td>
</tr>
</tbody>
</table>

Figure 3.4 Illustration of Timestamp Adjustment Based on User Clock
microseconds. The total seconds and microseconds are used for timestamp adjustments based on either relative or absolute time reference. In relative time reference with the system clock, the timestamp is added to the start time of the input script file. In absolute time reference, the timestamp is added to the start time of the NPSNET-OASIS system. The same procedure is also applied to the user clock reference.

B. STRUCTURE OF SCRIPT EVENTS

Each script event has a 3D icon identification, a position, a timestamp and attributes to describe a task-level behavior event. Users of NPSNET-OASIS must be familiar with the data structures and the information contained in them.

1. 3D Icon Identification

3D icon identifications include an object number, a host number, an object description, and an object status. Object numbers are four digit values assigned to 3D icons by the visual simulator to distinguish from other 3D icons. Host numbers are three digit values used in identifying visual simulators in a network. When combined, a host number and an object number provide a unique identification number to a 3D icon (Figure 3.5). Object descriptions are descriptive names of 3D icons. NPSNET requires them for associating OFF files for visual display. Supplementing world state information, object status is a description of the current state on a 3D icon.

```
501 1234
```

Host Number   Object Identification Number

Figure 3.5 The 3D Icon Identification Number
2. Event Position

Positions for a script event include grid coordinates and an elevation. Elevation, represented in meters, is the altitude of an area above sea level. The primary military grid reference system in the United States, the Universal Transverse Mercator (UTM), is a world wide plane coordinate system based on the metric standard [Ref. 4]. The grid coordinate system of UTM is adopted in NPSNET-OASIS (Figure 3.6). Each UTM grid zone is a square area of six degrees in longitude by eight degrees in latitude. UTM coordinates are designated by two or three characters. The last character, in alphabetic notation, represents the latitude offset, and the beginning characters, in numeric notation, represents the longitude offset. To further identify locations in each UTM grid zone, the U.S. Army created the MGRS [Ref. 4]. MGRS subdivides UTM grid zones into 100,000 meter square areas designated by two letters. To complete the MGRS grid, UTM easting and northing are used to designate which square meter area. Coordinates are in even digits where the first half representing easting, and the second half, northing. Precision in UTM coordinates requires five digits for easting and northing. In NPSNET-OASIS, easting and northing coordinates with digits less than five, are appended with additional zeros (Figure 3.7).

![Figure 3.6 UTM Coordinate Position](image-url)

<table>
<thead>
<tr>
<th>79A -DN -12345 12345</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gridzone</td>
</tr>
</tbody>
</table>

16
3. **Timestamp**

Timestamps are based upon the 24-hour clock metaphor where times are represented as strings [Ref. 1]. Each field of the timestamp string is represented by numeric characters with leading zeros being optional. In Figures 3.8a-c, hours, minutes, and seconds of the timestamp string are interpreted from right to left. Microseconds are interpreted from left to right (Figure 3.8). Used in event scheduling, timestamps indicate when to execute the event. In time delays, timestamps indicate the time durations of the script pauses, or the time to begin reading scripts. In Figure 3.8, timestamps are listed in formats acceptable to NPSNET-OASIS.

4. **Event Attributes**

Representing properties of continuous and transient events, attributes provide additional information on script events. Attributes enable a 3D icon to be unique among other 3D icons in simulation. Information such as headings and velocities of 3D icons are affected by state changes of continuous events. Appearances of 3D icons are affected by state changes of transient events. An example of a transient event is an explosion of a 3D icon. The attributes for this event require the description type and the bounding area of the explosion. See Appendix I for description of event attributes.

---

Figure 3.7 UTM Coordinate Position With Padded Zeros

<table>
<thead>
<tr>
<th>Gridzone</th>
<th>MGRS</th>
<th>Easting</th>
<th>Northing</th>
</tr>
</thead>
<tbody>
<tr>
<td>79A-DN</td>
<td>12300</td>
<td>12300</td>
<td></td>
</tr>
</tbody>
</table>

Appended Zeros
Figure 3.8 Timestamp Formats

C. SCRIPT OPTIONS

Script options, or directives, are used in script files for file operations, script system defaults, and assigning values for timestamp adjustments. From Figure 2.6, the script is modified to include all the script options available to NPSNET-OASIS (Figures 3.9 and 3.10). Syntax for each script option is listed in Appendix A.

1. Script_Abort

Normal termination of NPSNET-OASIS requires all statements in the input script file to be processed. However, the animator has the option to terminate the script by inserting
>Description: Sample script of two M-1 tanks 'script.ml'
Host Id Number: 501
Simulator: NPSNET on IRIS VGX Workstation
Author: Phillip West

******************************************************************************

/* Assign timestamp reference relative to start time of input script */
SCRIPT_TIMESTAMP relative

/* Assign default UTM gridzone and MGRS */
SCRIPT_LOCATION II8-DN

/* Assign time factor of 50 percent for all timestamps */
SCRIPT_TIME_FACTOR 0.5

/* Assign time adjustment value of 10 seconds for all timestamps */
SCRIPT_TIME_ADJUSTMENT 10.0

/* Activate clouds with velocity 5.0 km/h, heading 270.0 west */
ENVIRONMENT_ACTIVATE 5010500 Cloud 270.0 5.0 34536783 1000.0 30.0

/* Activate two M-1 tanks */
VEHICLE_ACTIVATE 5010001 M1 090.0 40.0 344955 0.0 40.0
VEHICLE_ACTIVATE 5010002 M1 090.0 40.0 345958 0.0 45.0

/* Change headings of M-1 tanks */
VEHICLE_HEADING 5010001 135.0 344801 0.0 2:0.0
VEHICLE_HEADING 5010002 120.0 344803 0.0 2:5.0

/* Change velocities of M-1 tanks */
VEHICLE_SPEED 5010001 5.0 344670 0.0 6:0.0
VEHICLE_SPEED 5010001 5.0 344677 0.0 7:25.0

/* M-1 tanks passing by a building */
OBJECT_ACTIVATE 5010004 Building 344701 0.0 8:10.0
OBJECT_DEACTIVATE 5010004 9:40.0

/* Script message for advance warning of upcoming script events */
SCRIPT_MESSAGE Activating column of jeeps

/* Script call for column of jeeps */
SCRIPT_CALL script-2.ml

/* Continue script with next script file */
SCRIPT_CHAIN script-1.ml

Figure 3.9 Sample Script With Script Options
Description: Sample script of two M-1 tanks 'script-1.ml'
Host Id Number: 501
Simulator: NPSNET on IRIS VGX Workstation
Author: Phillip West

************
/* M-1 tanks passing by a palm tree */
OBJECT_ACTIVATE 5010003 PalmTree 345700 0.0 22:0.0
OBJECT_DEACTIVATE 5010003 22:35.0

/* Change velocities of M-1 tanks */
VEHICLE_SPEED 5010001 5.0 345670 0.0 26:0.0
VEHICLE_SPEED 5010001 5.0 345677 0.0 27:25.0

/* Deactivate M-1 tanks */
VEHICLE_DEACTIVATE 5010001 29:45.0
VEHICLE_DEACTIVATE 5010002 29:50.0

/* Deactivate clouds */
ENVIRONMENT_ACTIVATE 5010500 30:0.0

/* Script delay for duration of 5 minutes prior to termination */
SCRIPT_DELAY 5:0.0 absolute

/* Write to output script file message for next script processing */
SCRIPT_FILE_WRITE SCRIPT_MESSAGE End of Script

/* Terminate script */
SCRIPT_ABORT

Description: Sample script of multiple jeeps 'script-2.ml'
Host Id Number: 501
Simulator: NPSNET on IRIS VGX Workstation
Author: Phillip West

************
/* Activate jeep vehicle */
VEHICLE_ACTIVATE 5010010 Jeep 090.0 40.0 345678 0.0 10:0.0

/* Repeat script for 10 iterations with total 11 separate vehicles */
SCRIPT_REPEAT 10 1 1:0.0

Figure 3.10 Sample Script With Script Options
SCRIPT_ABORT. This script directive has no arguments. When encountered, all script files are closed, and a script result of END_OF_SCRIPT is returned to the system.

2. Script_Call

The directive SCRIPT_CALL is a subscript call, similar to a procedure call in a high-level programming language. The argument for this directive is a string containing the script file name. The calling script file is suspended during processing of the subscript and control returns on subscript’s end of file. This directive supports modular scripting in NPSNET-OASIS.

3. Script.Chain

Linking of one input script file to another requires the script directive SCRIPT_CHAIN. The required argument is a string containing the script file name. After the chained script file is successfully opened, the other script file is closed. This directive is useful for combining small script files into one large script.

4. Script.Repeat

Similar to a counter-controlled loop, the script directive SCRIPT_REPEAT allows scripts to be processed repeatedly. There are three arguments for SCRIPT_REPEAT - iterations, object number adjustment, and timestamp adjustment. Iterations are values for the number of times repeating the same script. Object number adjustment is the increment/decrement value for all 3D icon identification numbers. Similar to the script option SCRIPT_TIME_ADJUSTMENT, timestamp adjustment is the increment/decrement of timestamps for each repeated script event. All script events in script chains and script calls are affected by SCRIPT_REPEAT. This directive is useful for repeating the entire script, or repeating a short series of script events.

5. Script.Timestamp

SCRIPT_TIMESTAMP, with the selected string as argument, is used to change time reference in script processing. The default time reference for NPSNET-OASIS system is
"relative". The other option is "absolute". Relative time reference is for all timestamp adjustments based on input script start time. Absolute time reference is for all timestamp adjustments based on system start time.

6. Script Time Adjustment

Changing event timestamps throughout the script requires SCRIPT_TIME_ADJUSTMENT to be inserted prior to selected statements. An argument in timestamp format, is used for adding to event timestamps. To stop timestamp adjustments, requires another SCRIPT_TIME_ADJUSTMENT and an argument of zero.

7. Script Time Factor

Another option for adjusting timestamp, is the use of SCRIPT_TIME_FACTOR. With a floating-type numeric argument, SCRIPT_TIME_FACTOR, provides the time factor percentage for each event timestamp. This option is useful for incrementing or decrementing speed of script processing. For example, the value of 2.0 causes two seconds of script time to be four seconds of wall clock time.

8. Script Delay

There are two arguments required for SCRIPT_DELAY - timestamp and delay type. Delay type is either "absolute" or "relative". In absolute mode, the current script is suspended until the time specified by the timestamp. In relative mode, the current script is suspended for the time duration specified by the timestamp.

9. Script Location

Upon initialization, the NPSNET-OASIS system assigns the default UTM gridzone and MGRS with strings "10S" and "DN" respectfully. Defaults will be assigned when an UTM gridzone and/or MGRS are not included in the event position. Replacing one or both system defaults requires SCRIPT_LOCATION to be used in the script.
10. **Script_File_Write**

Script objects recorded to an output script file include script events and script comments. There are no script options or blank lines. SCRIPT_FILE_WRITE provides animators the capability to include any type of script statements in a recorded script. If there is no output script file, then an error message is displayed.

11. **Script_Message**

While developing software, programmers include statements in their source code for tracing and debugging. In NPSNET-OASIS, animators have the same capability by allowing script messages to be displayed on a standard output device. Messages can include any text desired by the animator. One suggestion is to use script messages prior to selected script events. Messages with information on upcoming events provide advanced warning of what to expect in visual simulation.
IV. IMPLEMENTATION OF NPSNET-OASIS

A. OVERVIEW

The current NPSNET system runs across the entire Silicon Graphics, Inc. (SGI) IRIS 4D line [Ref. 10]. NPSNET-OASIS was developed with no graphics function calls, thus allowing application on various types of platforms including non-graphics workstations. Networking allows the different platforms to interact on other workstations in the Graphics and Video Laboratory.

For autonomous vehicle control, the network harness process of NPSNET enables NPSNET-OASIS to provide scripted autonomous players in visual simulation [Ref. 10]. The network harness uses Ethernet TCP/IP multicast packets designed for the NPSNET system. The purpose of this process is to listen to the packets broadcast on the network and to build an internal model of the state of the world from those packets. Script events from NPSNET-OASIS are transformed to message packets and broadcasted via the network harness. In addition, packets received are transformed and recorded to an output script.

B. DATA STRUCTURES

The data structures required for the script event and message packet transformations are contained in Appendices I through K. Currently in NPSNET, only vehicle script events are used for autonomous vehicle control. As further development of NPSNET continues, other script events are integrated to allow enhanced modeling of the world.

C. NPSNET-OASIS NETWORK INTERFACE

As shown in Figure 4.1, NPSNET-OASIS is integrated into NPSNET by coupling with the NPSNET-OASIS network interface. The network interface provides the
synchronization in playbacks and recordings of NPSNET-OASIS scripts. Using the required data structures, transformations of all script event message packets are processed in the network interface.

1. Internal Data Structures

Several NPSNET-OASIS network interfaces can interact with NPSNET with each containing its own internal data structures for representing the state of the world. Referred to as the local state of the world, the data structure contains all locations and vehicle types
for NPSNET-OASIS. Additionally, another internal data structure is used for indicating which vehicle icons are recorded on script. First time script events of vehicle icons require the script event VEHICLE_ACTIVATE to be recorded on script. Playback of scripts requires VEHICLE_ACTIVATE to set the vehicle attribute alive for the vehicle icon to be active in NPSNET. The network interface updates the local data structures prior to each message broadcast and prior to recording on script.

2. Message Packets

Currently there is only one type of message packet used in NPSNET for scripted autonomous players - vehicle update. Each vehicle script event is processed as a vehicle update regardless of event type. The encoding of the message packet is described in Appendix K.

There are three types of message packets received from the network - stop script, time synchronization, and vehicle update. Stop script message enables the NPSNET-OASIS network interface to detach from the network. Time synchronization message assigns the initial start time of NPSNET. NPSNET-OASIS sets all start times of script files and script system to the time specified in the message. Vehicle update message is used to update the local state of the world and to record the script event.

3. Operation of Network Interface

Referring to Figure 4.2, the NPSNET-OASIS network interface is operational until end of input script file or a stop script message packet. If the network interface was initialized with no input script file, then a stop script message is required. Prior to reading script events, the network is checked for incoming message packets. When message packets exist, the local state of the world is updated and the transformed script events are recorded. All script events from the input script file are transformed to message packets.
and sent to the network. System time for NPSNET-OASIS is based on the user clock reference.
void MonitorNpsnetNetwork() {
    AddProcessToNetwork();
    SetSystemStartTime(StartTicks / HZ, StartTicks % HZ);
    while(ActiveNetworkInterface) {
        CurrentTicks = (times(&SysTimes) - StartTicks);
        ReceiveAndProcessMessages();
        if(InputScriptExists && ActiveNetworkInterface) {
            ReadAndProcessScriptEvent(CurrentTicks, VehicleArray);
            switch(GetScriptResult()) {
                case END_OF_SCRIPT:
                    InputScriptExists = FALSE;
                    if(!OutputScriptExists)
                        ActiveNetworkInterface = FALSE;
                    break;
                case SCRIPT_VALID:
                    WriteScriptMessage(GetIconID(), GetEventID());
                    SendUpdateMessage(GetIconID());
                    break;
                case SCRIPT_ERROR:
                case SCRIPT_IN_DELAY:
                case NO_SCRIPT_EVENT:
                    default:
                        break;
            }
        } // endif
    } // endloop
    DetachProcessFromNetwork();
} // end MonitorNpsnetNetwork

Figure 4.2 Network Interface Monitor Routine
V. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY AND CONCLUSION

The NPSNET-OASIS system has fulfilled the initial requirement of providing tools to record and playback scripts. The NPSNET-OASIS network interface provides users of NPSNET the capability to generate scripted autonomous players in visual simulations. The study in applying object-oriented methodologies in a practical design such as NPSNET-OASIS was beneficial for extensibility and reusability of existing source code. This was proven when the network interface for NPSNET-OASIS was built on top of a script system. As new software developments evolve in the Graphics and Video Laboratory, changes can be made in NPSNET-OASIS with little modifications.

B. LIMITATIONS

There are two limitations in NPSNET-OASIS. First, using the classes of NPSNET-OASIS requires C++ throughout software development. Existing software in the Graphics and Video Laboratory such as NPSNET must be converted to C++ in order to fully integrate NPSNET-OASIS in code. The second limitation is that NPSNET is currently unable to use all the script events in NPSNET-OASIS. The data structures for NPSNET are limited in containing the information for all types of continuous and transient events. In addition, the capabilities for transient events such as collisions and explosions are still under development. Current design of the NPSNET-OASIS network interface permits only continuous events of vehicle icons, such as vehicle heading and velocity changes.

C. FUTURE DIRECTIONS

Current implementations of NPSNET-OASIS are in executable load modules with no graphics functions. Execution of load module requires typing the module name at the command prompt. There are no output displays other than error messages generated during
processing and recording of scripts. Future effort could be placed on developing some interactive tool for using the script tools of NPSNET-OASIS, such as a graphics script editor or a syntax directed editor.

The NPSNET-OASIS network interface is implemented only for the SGI IRIS graphics workstations. Porting the software to other platforms should be an easy matter. Suggestion is to modify the source code of NPSNET-OASIS for providing scripted autonomous players from a different platform.
APPENDIX A
NPSNET-OASIS Script Events, Results, and Errors

Note:  Back slashes \ in syntax or examples indicate continuation on the same line.

**Vehicle Events**

**VEHICLE_ACTIVATE**

```
VEHICLE_ACTIVATE <vehicle number> <type description> <heading> \ 
<velocity> <UTM coordinates> <elevation> \ 
<timestamp>
```

Description - New vehicle icon activated for visual simulation.

Example -  `VEHICLE_ACTIVATE 200 M1 235.0 5.0 1234567890 0.0 5:0.0`

**VEHICLE_DEACTIVATE**

```
VEHICLE_DEACTIVATE <vehicle number> <timestamp>
```

Description - Vehicle icon deactivated from visual simulation.

Example -  `VEHICLE_DEACTIVATE 200 5:0.0`

**VEHICLE_MODIFICATION**

```
VEHICLE_MODIFICATION <vehicle number> <component name> \ 
<rotation X direction> \ 
<rotation Y direction> <rotation Z direction> \ 
<translation X direction> \ 
<translation Y direction> \ 
<translation Z direction> <UTM coordinates> \ 
<elevation> <timestamp>
```

Description - Vehicle icon is modified by changing the degrees of freedom on one of it's components.
Example - VEHICLE Modification

200 turret 10.0 10.0 10.0 25.0 25.0 25.0 1234567890 0.0 5:0.0

**VEHICLE_PITCH**

VEHICLE_PITCH  

Example - VEHICLE_PITCH 200 10.0 1234567890 0.0 5:0.0

**VEHICLE_ROLL**

VEHICLE_ROLL  

Example - VEHICLE_ROLL 200 10.0 stbd 1234567890 0.0 5:0.0  
VEHICLE_ROLL 200 10.0 port 1234567890 0.0 5:0.0  
VEHICLE_ROLL 200 0.0 none 1234567890 0.0 5:0.0

**VEHICLE_POSITION**

VEHICLE_POSITION  

Example - VEHICLE_POSITION 200 79A-DN-1234567890 0.0 5:0.0  
VEHICLE_POSITION 200 79A-1234567890 0.0 5:0.0  
VEHICLE_POSITION 200 DN-1234567890 0.0 5:0.0
VEHICLE_HEADING

VEHICLE_HEADING  <vehicle number>  <heading>  <UTM coordinates>  \
<elevation>  <timestamp>

Description - The heading of vehicle icon is changed. All headings are from 0.0 to 359.9 degrees relative to North. For NPSNET, degrees are required to be in radians.

Example -  VEHICLE_HEADING 200 270.0 79A-DN-1234567890 0.0 5:0.0

VEHICLE_SPEED

VEHICLE_SPEED  <vehicle number>  <velocity>  <UTM coordinates>  \
<elevation>  <timestamp>

Description - The velocity of vehicle icon is changed. All velocities are in kilometers per hour.

Example -  VEHICLE_SPEED 200 10.0 79A-DN-1234567890 0.0 5:0.0

VEHICLE_TRAIL

VEHICLE_TRAIL  <vehicle number>  <trail description>  \
<bound area X direction>  \n<bound area Y direction>\n<bound area Y direction>  <offset X direction>  \n<offset Y direction>  <offset Z direction>  \n<UTM coordinates>  <elevation>  <timestamp>

Description - The trail of vehicle icon is activated for visual simulation. Trails such as dust or wake requires attributes for bounding area for size and vehicle offset.

Example -  VEHICLE_TRAIL 200 mediumdust 10.0 10.0 10.0 0.0 0.0 0.0 0.0 \n1234567890 0.0 5:0.0

VEHICLE_COLLISION

VEHICLE_COLLISION  <vehicle number>  <description result>  \
<UTM coordinates>  <elevation>  <timestamp>
Description - This event is a transient event indicating vehicle icon collision with another icon.

Example - VEHICLE_COLLISION 200 destroyed 79A-DN-1234567890 0.0 5:0.0

VEHICLE_EXPLOSION

VEHICLE_EXPLOSION     <vehicle number> <explosion description> \\
<bound area X direction> \\
<bound area Y direction> \\
<bound area Y direction> <UTM coordinates> \\
<elevation> <timestamp>

Description - This event is a transient event of a vehicle icon explosion. Attributes required are explosion type and bounding area for explosion.

Example - VEHICLE_EXPLOSION 200 largescale 10.0 10.0 10.0 1234567890 \\
0.0 5:0.0

VEHICLE_FLAMING

VEHICLE_FLAMING     <vehicle number> <description of fire> \\
<bound area X direction> \\
<bound area Y direction> \\
<bound area Y direction> <offset X direction> \\
<offset Y direction> <offset Z direction> \\
<UTM coordinates> <elevation> <timestamp>

Description - This event is a transient event of a vehicle icon on fire. Attributes required are type of flames, bounding area of the flames, and the vehicle offset to the flames.

Example - VEHICLE_FLAMING 200 mediumflame 10.0 10.0 10.0 1234567890 \\
0.0 5:0.0

VEHICLE_SMOKING

VEHICLE_SMOKING     <vehicle number> <description of smoke> \\
<bound area X direction> \\
<bound area Y direction> \\
<bound area Y direction> <offset X direction> \\
<offset Y direction> <offset Z direction> \\
<UTM coordinates> <elevation> <timestamp>
Description - This event is a transient event of a vehicle icon emitting smoke. Attributes required are smoke type, bounding area of the smoke, and the vehicle offset to the smoke.

Example - VEHICLE_SMOKING 200 mediumsmoke 10.0 10.0 0.0 0.0 0.0 0.0 1234567890 0.0 5:0.0

VEHICLE_DESTROYED

VEHICLE_Destroyed <vehicle number> <UTM coordinates> <elevation> <timestamp>

Description - Not removed from visual simulation, the vehicle icon is destroyed.

Example - VEHICLE_DESTROYED 1234567890 0.0 5:0.0

VEHICLE_FUEL

VEHICLE_FUEL <vehicle number> <fuel amount> <timestamp>

Description - The fuel state is changed on a vehicle icon. Amount of fuel left in vehicle icon is indicated in liters.

Example - VEHICLE_FUEL 200 10.0 5:0.0

VEHICLE_AMMUNITION

VEHICLE_AMMUNITION <vehicle number> <weapon name> <weapon rounds> <timestamp>

Description - The ammunition state is changed on a vehicle icon. Weapon rounds of weapon name indicate current number assigned to the vehicle icon.

Example - VEHICLE_AMMUNITION 200 turret 10 5:0.0

VEHICLE_UPDATE

VEHICLE_UPDATE <vehicle number> <vehicle status> <heading> <velocity> <pitch angle> <roll angle> <roll direction> <UTM coordinates> <elevation> <timestamp>
Description - Update status on a vehicle icon.

Examples - VEHICLE_UPDATE 200 inmotion 260.0 10.0.0 0.0 none 456789\0.0 5:0.0

VEHICLE_STATUS_QUERY

VEHICLE_STATUS_QUERY <vehicle number> <timestamp>

Description - This event queries status of a selected vehicle icon.

Example - VEHICLE_STATUS_QUERY 200 5:0.0

Weapon Events

WEAPON_LAUNCH

WEAPON_LAUNCH <weapon number> <source id> <description>\<heading> <heading type> <velocity> \<pitch angle> <UTM coordinates> <elevation> \<timestamp>

Description - Weapon is launch from a source vehicle icon with initial velocity and headings. Headings are indicated in relative 'R' or absolute 'A'.

Examples - WEAPON_LAUNCH 300 200 sm-2 260.0 A 500.9 45.0 456789 3.0\5:0.0
WEAPON_LAUNCH 300 200 sm-2 260.0 R 500.9 45.0 456789 3.0\5:0.0

WEAPON_IMPACT

WEAPON_IMPACT <weapon number> <source id> <UTM coordinates>\<elevation> <timestamp>

Description - Weapon impact terminating weapon icon.

Example - WEAPON_IMPACT 300 200 456789 3.0 5:0.0
WEAPON_EXPLOSION

```
WEAPON_EXPLOSION <weapon number> <source id> <UTM coordinates> \ 
<elevation> <timestamp>
```

Description - Weapon explosion terminating weapon icon.

Example - `WEAPON_EXPLOSION 300 200 456789 3.0 5:0.0`

WEAPON_UPDATE

```
WEAPON_UPDATE <weapon number> <source id> <weapon status> \ 
<heading> <heading type> <velocity> \ 
<pitch angle> <UTM coordinates> <elevation> \ 
<timestamp>
```

Description - Update status on a weapon icon.

Examples - 
```
WEAPON_UPDATE 300 200 active 260.0 A 300.0 35.0 456789 3.0 5:0.0

WEAPON_UPDATE 300 200 active 260.0 R 300.0 35.0 456789 3.0 5:0.0
```

WEAPON_STATUS_QUERY

```
WEAPON_STATUS_QUERY <weapon number> <source id> <timestamp>
```

Description - This event queries status of a selected weapon icon.

Examples - `WEAPON_STATUS_QUERY 300 200 5:0.0`

Miscellaneous Object Events

OBJECT_ACTIVATE

```
OBJECT_ACTIVATE <object number> <object description> \ 
<UTM coordinates> <elevation> <timestamp>
```

Description - New miscellaneous object icon activated for visual simulation.

Example - `OBJECT_ACTIVATE 250 building 1234567890 5:0.0`
OBJECT_DEACTIVATE

OBJECT_DEACTIVATE <object number> <timestamp>

Description - Miscellaneous object icon deactivated from visual simulation.

Example - OBJECT_DEACTIVATE 250 5:0.0

OBJECT_COLLISION

OBJECT_COLLISION <object number> <description result> \ 
<UTM coordinates> <elevation> <timestamp>

Description - This event is a transient event indicating miscellaneous object icon collision with another icon.

Example - OBJECT_COLLISION 250 destroyed 79A-DN-1234567890 0.0 5:0.0

OBJECT_EXPLOSION

OBJECT_EXPLOSION <object number> <explosion description> \ 
<bound area X direction> \ 
<bound area Y direction> \ 
<bound area Y direction> <UTM coordinates> \ 
<elevation> <timestamp>

Description - This event is a transient event of a miscellaneous object icon explosion. Attributes required are explosion type and bounding area for explosion.

Example - OBJECT_EXPLOSION 250 largescale 10.0 10.0 10.0 12345678 0.0 5:0.0

OBJECT_FLAMING

OBJECT_FLAMING <object number> <description of fire> \ 
<bound area X direction> \ 
<bound area Y direction> \ 
<bound area Y direction> <offset X direction> \ 
<offset Y direction> <offset Z direction> \ 
<UTM coordinates> <elevation> <timestamp>

38
Description - This event is a transient event of a miscellaneous object icon in flames. Attributes required are flame type, bounding area of the smoke, and the object offset of the flames.

Example -  
```
OBJECT_FLAMING 250 largescale 10.0 10.0 10.0 0.0 0.0 0.0
1234567890 0.0 5:0.0
```

OBJECT_SMOKING

```
OBJECT_SMOKING  
<object number> <description of smoke> \  
<bound area X direction> \  
<bound area Y direction> \  
<bound area Y direction> <offset X direction> \  
<offset X direction> <offset Z direction> \  
<U TM coordinates> <elevation> <timestamp>
```

Description - This event is a transient event of a miscellaneous object icon emitting smoke. Attributes required are smoke type, bounding area of the smoke, and the object offset of the smoke.

Example -  
```
OBJECT_SMOKING 250 mediumsmoke 10.0 10.0 10.0 0.0 0.0 0.0
1234567890 0.0 5:0.0
```

OBJECT_UPDATE

```
OBJECT_UPDATE  
<object number> <object status> \  
<U TM coordinates> <elevation> <timestamp>
```

Description - This event queries status of a selected miscellaneous object icon.

Examples -  
```
OBJECT_UPDATE 250 active 1234567890 0.0 5:0.0
```

OBJECT_STATUS_QUERY

```
OBJECT_STATUS_QUERY  
<object number> <timestamp>
```

Description - This event queries status of a selected miscellaneous object icon.

Example -  
```
OBJECT_STATUS_QUERY 250 5:0.0
```
Environment Events

ENVIRONMENT_ACTIVATE

ENVIRONMENT_ACTIVATE  <environment object number>  <type description>  \\  <heading>  <velocity>  <UTM coordinates>  \\  <elevation>  <timestamp>

Description - New environment effect icon is activated for visual simulation.
Example -   ENVIRONMENT_ACTIVATE  400 cloud 345.0 5.0 1234567890 5:0.0

ENVIRONMENT_DEACTIVATE

ENVIRONMENT_DEACTIVATE  <environment object number>  <timestamp>

Description - Environment effect icon of vehicle number is deactivated from visual simulation.
Example -   ENVIRONMENT_DEACTIVATE  400 5:0.0

Script Options

SCRIPT_ABORT

SCRIPT_ABORT

Description - Normal termination of NPSNET-OASIS requires all statements in the input script file to be processed. However, the animator has the option to terminate the script by inserting SCRIPT_ABORT. This script directive has no arguments. When encountered, all script files are closed, and a script result of END_OF_SCRIPT is returned to the system.
Example -  SCRIPT_ABORT

SCRIPT_CALL

SCRIPT_CALL  <filename for script call>

Description - The directive SCRIPT_CALL is a subscript call, similar to a procedure call
in a high-level programming language. The argument for this directive is a string containing the script file name. The calling script file is suspended during processing of the subscript. Only one script is processed at any given time. Script file control returns to the caller on subscript's end of file. This directive supports modular scripting in NPSNET-OASIS.

Example - `SCRIPT_CALL script-1.ml`

**SCRIPT_CHAIN**

`SCRIPT_CHAIN <filename for script chaining>`

**Description** - Linking of one input script file to another requires the script directive `SCRIPT_CHAIN`. The required argument is a string containing the script file name. After the chained script file is successfully opened, the other script file is closed. This directive is useful for combining small script files into one large script.

Example - `SCRIPT_CHAIN script-1.ml`

**SCRIPT_REPEAT**

`SCRIPT_REPEAT <iterations> <object number adjustment> \ <time adjustment>`

Similar to a counter-controlled loop, the script directive `SCRIPT_REPEAT` allows scripts to be processed repeatedly. There are three arguments for `SCRIPT_REPEAT` - iterations, object number adjustment, and timestamp adjustment. Iterations are values for the number of times repeating the same script. Object number adjustment is the increment/decrement value for all 3D icon identification numbers. Similar to the script option `SCRIPT_TIME_ADJUSTMENT`, timestamp adjustment is the increment/decrement of timestamps for each repeated script event. All script events in script chains and script calls are affected by `SCRIPT_REPEAT`. This directive is useful for repeating the entire script, or repeating a short series of script events.
Example - `SCRIPT_REPEAT 10 1 2:0.0`

**SCRIPT_LOCATION**

```
SCRIPT_LOCATION  
<new default UTM gridzone and/or MGRS>
```

**Description** - Upon initialization, the NPSNET-OASIS system assigns the default UTM gridzone and MGRS with strings “10S” and “DN” respectfully. Defaults will be assigned when an UTM gridzone and/or MGRS are not included in the event position. Replacing one or both system defaults, requires `SCRIPT_LOCATION` to be used in the script.

**Examples** - `SCRIPT_LOCATION 79A-DN`

```
SCRIPT_LOCATION 79A
SCRIPT_LOCATION DN
```

**SCRIPT_TIMESTAMP**

```
SCRIPT_TIMESTAMP  
<time reference for script timestamps>
```

**Description** - The default time reference for NPSNET-OASIS system is “relative”. The other option is “absolute”. `SCRIPT_TIMESTAMP`, with the selected string as argument, is used to change time reference in script processing. Relative time reference is for all timestamp adjustments based on input script start time. Absolute time reference is for all timestamp adjustments based on system start time.

**Examples** - `SCRIPT_TIMESTAMP relative`

```
SCRIPT_TIMESTAMP absolute
```

**SCRIPT_TIME_FACTOR**

```
SCRIPT_TIME_FACTOR  
<time factor value>
```

**Description** - Another option for adjusting timestamp, is the use of `SCRIPT_TIME_FACTOR`. With a floating-type numeric argument, `SCRIPT_TIME_FACTOR`, provides the time factor percentage for each event timestamp. This option is useful for incrementing or decrementing speed of script processing.
**Example -** `SCRIPT_TIME_FACTOR 0.5`

**SCRIPT_TIME_ADJUSTMENT**

`SCRIPT_TIME_ADJUSTMENT <timestamp>`

**Description** - Changing event timestamps throughout the script requires `SCRIPT_TIME_ADJUSTMENT` to be inserted prior to selected statements. An argument in timestamp format, is used for adding to event timestamps. To stop timestamp adjustments, requires another `SCRIPT_TIME_ADJUSTMENT` and an argument of zero.

**Example -** `SCRIPT_TIME_ADJUSTMENT 10:0.5`

**SCRIPT_DELAY**

`SCRIPT_DELAY <timestamp> <type of script delay>`

**Description** - There are two arguments required for `SCRIPT_DELAY` - timestamp and delay type. Delay type is either "absolute" or "relative". In absolute, the current script is suspended until the time specified by the timestamp. In relative, the current script is suspended for the time duration specified by the timestamp.

**Examples -**

- `SCRIPT_DELAY 10:0.0 relative`
- `SCRIPT_DELAY 25:24.0 absolute`

**SCRIPT_FILE_WRITE**

`SCRIPT_FILE_WRITE <script line>`

**Description** - Script objects recorded to an output script file include script events and script comments. There are no script options or blank lines. `SCRIPT_FILE_WRITE` provides animators the capability to include any type of script statements in a recorded script. If there is no output script file, then an error message is displayed.

**Example -** `SCRIPT_FILE_WRITE SCRIPT_MESSAGE end of script`
**SCRIPT_MESSAGE**

**SCRIPT_MESSAGE** <message>

**Description** - While developing software, programmers include statements in their source code for tracing and debugging. In NPSNET-OASIS, animators have the same capability by allowing script messages to be displayed on a standard output device. Messages can include any text desired by the animator. One suggestion is to use script messages prior to selected script events. Messages with information on upcoming events provide advanced warning of what to expect in visual simulation.

**Example** - SCRIPT_MESSAGE End Of Script
Script Results

SCRIPT_ERROR

SCRIPT_ERROR is returned from NPSNET-OASIS when an input/output file error occurred or a script statement is invalid. File errors occur if unable to open or close a file. Invalid script statements include invalid syntax for script comments, invalid procedural operators, invalid arguments for a procedural operator, and invalid formats for arguments.

END_OF_SCRIPT

On end of script, which includes processing of all input script files, a result of END_OF_SCRIPT is returned.

SCRIPT_IN_DELAY

SCRIPT_IN_DELAY is returned when processing of script files are suspended.

SCRIPT_VALID

For a result of SCRIPT_VALID, script statements are of valid syntax.

NO_SCRIPT_EVENT

An option for classes built on top of NPSNET-OASIS, NO_SCRIPT_EVENT is returned when script processing is suspended or a script event is not ready for return. Used in the NPSNET-OASIS network interface, NO_SCRIPT_EVENT is returned when the script event is not ready for multicast.
**Script Errors**

NPSNET-OASIS can only have one main input script file assigned. If another is assigned then the following is displayed:

`SCRIPT ERROR: Only one main script permitted to be open.`

The following message is displayed if the main script file does not exist. Possible errors can include file does not exist, or some characters of the file name are of upper case.

`SCRIPT ERROR: Unable to open main script file 'filename'`.  

When NPSNET-OASIS is unable to close a script file a message is displayed.

`SCRIPT ERROR: Unable to close 'filename'`.

There are error messages for invalid file opens for script chaining or script call. In addition, another line is displayed informing the line number and file name of the script file containing the script option. Messages are as follows:

`SCRIPT ERROR: Unable to open script 'filename' for chaining.`

`SCRIPT ERROR: Unable to open script 'filename' for script call`.

Script events to be recorded require an output script file. If the file does not exist, then the following error message is displayed:

`SCRIPT ERROR: No output file opened for script generation`.

For recording scripts, the following message is displayed if a procedural operator of a script event does not exist.

`SCRIPT ERROR: Invalid event for script generation`.  

46
If the script statement is not a script comment or a valid script procedural operator, then the following message is displayed:

*SCRIPT ERROR: Invalid OASIS Procedural Operator.*

For a valid script comment, the start symbol '/*' and the termination symbol '*/' must exist. In addition, no nesting of comments are permitted. When the termination symbol is encountered by the script processor, the rest of the script line is checked for invalid characters. Messages are as follows:

*SCRIPT ERROR: Invalid characters after script comment end symbol.*
*SCRIPT ERROR: Encountered another script comment beginning.*
*SCRIPT ERROR: Script comment with no end symbol.*

When script statements contain valid procedural operator, chances are that required arguments are missing, some arguments are of invalid format, or too many arguments. If this should happen, then an error message is displayed followed by another line containing line number and name of input script file. There is a separate error message for each procedural operator. Messages are as follows:

*SCRIPT ERROR: Invalid parameters for environment_activate.*
*SCRIPT ERROR: Invalid parameters for environment_deactivate.*
*SCRIPT ERROR: Invalid parameters for vehicle_activate.*
*SCRIPT ERROR: Invalid parameters for vehicle_deactivate.*
*SCRIPT ERROR: Invalid parameters for vehicle_modification.*
*SCRIPT ERROR: Invalid parameters for vehicle_pitch.*
*SCRIPT ERROR: Invalid parameters for vehicle_roll.*
*SCRIPT ERROR: Invalid parameters for vehicle_position.*
*SCRIPT ERROR: Invalid parameters for vehicle_heading.*
*SCRIPT ERROR: Invalid parameters for vehicle_speed.*
*SCRIPT ERROR: Invalid parameters for vehicle_trail.*
*SCRIPT ERROR: Invalid parameters for vehicle_collision.*
SCRIPT ERROR: Invalid parameters for vehicle Explosion.
SCRIPT ERROR: Invalid parameters for vehicle flaming.
SCRIPT ERROR: Invalid parameters for vehicle smoking.
SCRIPT ERROR: Invalid parameters for vehicle destroyed.
SCRIPT ERROR: Invalid parameters for vehicle fuel.
SCRIPT ERROR: Invalid parameters for vehicle ammunition.
SCRIPT ERROR: Invalid parameters for vehicle update.
SCRIPT ERROR: Invalid parameters for vehicle status query.
SCRIPT ERROR: Invalid parameters for object activate.
SCRIPT ERROR: Invalid parameters for object deactivate.
SCRIPT ERROR: Invalid parameters for object collision.
SCRIPT ERROR: Invalid parameters for object explosion.
SCRIPT ERROR: Invalid parameters for object flaming.
SCRIPT ERROR: Invalid parameters for object smoking.
SCRIPT ERROR: Invalid parameters for object update.
SCRIPT ERROR: Invalid parameters for object status query.
SCRIPT ERROR: Invalid parameters for script time factor.
SCRIPT ERROR: Invalid parameters for script time adjustment.
SCRIPT ERROR: Invalid parameters for script delay.
SCRIPT ERROR: Invalid parameters for script abort.
SCRIPT ERROR: Invalid parameters for script call.
SCRIPT ERROR: Invalid parameters for script chain.
SCRIPT ERROR: Invalid parameters for script repeat.
SCRIPT ERROR: Invalid parameters for script timestamp.
SCRIPT ERROR: Invalid parameters for script location.
SCRIPT ERROR: Invalid parameters for weapon launch.
SCRIPT ERROR: Invalid parameters for weapon impact.
SCRIPT ERROR: Invalid parameters for weapon explosion.
SCRIPT ERROR: Invalid parameters for weapon update.
SCRIPT ERROR: Invalid parameters for weapon status query.
APPENDIX B
NPSNET-OASIS Sample Scripts

FILE: script.ml

/* Assign timestamp reference relative to start time of input script */
SCRIPT_TIMESTAMP relative

/* Assign default UTM gridzone and MGRS */
SCRIPT_LOCATION 11S-DN

/* Assign time factor of 50 percent for all timestamps */
SCRIPT_TIME_FACTOR 0.5

/* Assign time adjustment value of 10 seconds for all timestamps */
SCRIPT_TIME_ADJUSTMENT 10.0

/* Activate clouds with velocity 5.0 km/h, heading 270.0 west */
ENVIRONMENT_ACTIVATE 5010500 Cloud 270.0 5.0 34536783 1000.0 30.0

/* Activate two M-1 tanks */
VEHICLE_ACTIVATE 5010001 M1 090.0 40.0 344955 0.0 40.0
VEHICLE_ACTIVATE 5010002 M1 090.0 40.0 345958 0.0 45.0

/* Change headings of M-1 tanks */
VEHICLE_HEADING 5010001 135.0 344801 0.0 2:0.0
VEHICLE_HEADING 5010002 120.0 344803 0.0 2:5.0

/* Change velocities of M-1 tanks */
VEHICLE_SPEED 5010001 5.0 344670 0.0 6:0.0
VEHICLE_SPEED 5010002 5.0 344677 0.0 7:25.0

/* M-1 tanks passing by a building */
OBJECT_ACTIVATE 5010004 Building 344701 0 8:10.0
OBJECT_DEACTIVATE 5010004 9:40.0

/* Script message for advance warning of upcoming script events */
SCRIPT_MESSAGE Activating column of jeeps

/* Script call for column of jeeps */
SCRIPT_CALL script-2.jeeps
/* Continue script with next script file */
SCRIPT_CHAIN script-1.ml

/********************
File: script-1.ml
Description: Sample script of two M-1 tanks in motion
Host Id Number: 501
Simulator: NFSNET on IRIS VGX Workstation
Author: Phillip West
********************/

/* M-1 tanks passing by a palm tree */
OBJECT_ACTIVATE 5010003 PalmTree 345700 0 22:0.0
OBJECT_DEACTIVATE 5010003 22:35.0

/* Change velocities of M-1 tanks */
VEHICLE_SPEED 5010001 5.0 345670 0.0 26:0.0
VEHICLE_SPEED 5010001 5.0 345677 0.0 27:25.0

/* Deactivate M-1 tanks */
VEHICLE_DEACTIVATE 5010001 29:45.0
VEHICLE_DEACTIVATE 5010002 29:50.0

/* Deactivate column of jeeps */
SCRIPT_CALL script-3.jeeps

/* Continue script with another scenario */
SCRIPT_CALL script-4.f14

/* Deactivate clouds */
ENVIRONMENT_DEACTIVATE 5010500 45:0.0

/* Script delay for duration of 2 minutes prior to termination */
SCRIPT_DELAY 2:0.0 absolute

/* Write to output script file message for next script processing */
SCRIPT_FILE_WRITE SCRIPT_MESSAGE End of Script

/* script message for end of script */
SCRIPT_MESSAGE *** DEMO COMPLETE ***

/* Terminate script */
SCRIPT_ABORT
/*********************************************************************/
File: script-2.jeeps
Description: Sample script of activating multiple jeeps
Host Id Number: 501
Simulator: NPSNET on IRIS VGX Workstation
Author: Phillip West
*******************************************************************************/

/* Activate jeep vehicle */
VEHICLE_ACTIVATE 5010010 Jeep 090.0 40.0 345678 0.0 10:0.0

/* Repeat script for 10 iterations with total 11 separate vehicles */
SCRIPT_REPEAT 10 1 1:0.0

*******************************************************************************/

/* Deactivate jeep vehicle */
VEHICLE_DEACTIVATE 5010010 29:0.0

/* Repeat script for 10 iterations with total 11 deactivate jeeps */
SCRIPT_REPEAT 10 1 1.0

51
/* Activate F-14 aircraft in flight status */
VEHICLE_ACTIVATE 5010110 F14 090.0 540.0 345678 1000.0 30:0.0

/* Aircraft roll to port 10 degrees */
VEHICLE_ROLL 5010110 10.0 port 344677 1000.0 31:10.0

/* Aircraft level roll to 0 degrees */
VEHICLE_ROLL 5010110 0.0 none 344676 996.7 34:56.5

/* Sea sparrow launch from F-14 */
WEAPON_LAUNCH 5010210 5010110 seasparrow 240.0 R 400.0 -10.0
344675 995.6 36:0.0

/* Weapon impact on surface */
WEAPON_IMPACT 5010210 5010110 342670 0.0 38:43.5

/* Weapon exploded on surface */
WEAPON_EXPLOSION 5010210 5010110 342670 0.0 39:0.0

/* Deactivate F-14 */
VEHICLE_DEACTIVATE 5010110 43:0.0
APPENDIX C

Class Definition of ScriptProcessor

// Classname: ScriptProcessor
// Derived from: VehicleProcessor, WeaponProcessor,
// MiscObjectProcessor, EnvironmentProcessor,
// ScriptOptionProcessor
// Base for: OasisSystem
// Remarks: This class provides the mechanism to read script
// files and process script events for valid script
// objects and comments.
//----------------------------------------------------------------------
class ScriptProcessor : private VehicleProcessor,
private WeaponProcessor,
private MiscObjectProcessor,
private EnvironmentProcessor,
private ScriptOptionProcessor {

public:
ScriptProcessor(); // Constructor
~ScriptProcessor(); // Destructor

// Member functions for ScriptProcessor
long ReadScriptObject(ScriptObject& Script);
// Reads next script object from input script file and
// returns null if error occurs.
SourceScriptFile* GetScriptFile();
void SetScriptFile(SourceScriptFile* Scriptfile);
// Assigns current script file for script processing.
void SetUtmScriptDefaults(char *Gridzone, char *Mgrs);
// Assigns default gridzones and mgrs for event positions.
ScriptProcessor* Instance();
// Returns reference pointer of instance.
virtual char *ClassName();
// Returns class identification string.

protected:
SourceScriptFile* CurrentScriptFile;
// Current input script file.

private:
long SetupScriptLine(char *Line);
// Removes excess blank spaces and converts uppercase
// characters to lower case. Returns SCRIPT_COMMENT
// if script line is the beginning of a comment block.
long ScanForBlankLines(char *Line);
// Returns null for non-blank lines.
long DetermineEvent(ScriptObject& Script, char* Line);
// Determines procedural operator for script object, and
// calls group processor for procedural operator. Returns
// null if script error occurs.
}); // end class ScriptProcessor
APPENDIX D
Class Definition of ScriptGenerator

// Class type: ScriptGenerator
// Derived from: OasisSystemObject
// Base for: OasisSystem
// Remarks: This class provides the mechanism in writing script files.

class ScriptGenerator : public OasisSystemObject {
public:
    ScriptGenerator(); // Constructor.
    ~ScriptGenerator(); // Destructor.

    // Member functions for ScriptGenerator
    long OpenOutputFile(char *Name); // Opens output script file and returns TRUE for successful file open.
    long WriteScriptObject(ScriptObject& Script); // Writes script object to script file and returns TRUE for successful file write.
    long WriteScriptLine(char *Line); // Writes script object to script file and returns TRUE for successful file write.
    long CloseOutputFile(); // Closes output script file and returns result of file close.
    ScriptGenerator* Instance(); // Returns reference pointer of instance.
    virtual char *ClassName(); // Returns class identification string.

protected:
    DestinationScriptFile OutFile; // Current destination script file.

private:
    void WriteVehicleEvent(VehicleEvent& Vehicle); // Writes script object of vehicle event to output script file.
    void WriteWeaponEvent(WeaponEvent& Weapon); // Writes script object of weapon event to output script file.
    void WriteMiscObjectEvent(MiscObjectEvent& MiscObject); // Writes script object of miscellaneous object event to output script file.
    void WriteEnvironmentEvent(EnvironmentEvent& Environment); // Writes script object of environment event to output script file.
};
void WriteComment(ScriptComment& Comment);
    // Writes script comment block to output script file.
}; // end class ScriptGenerator
APPENDIX E
Class Definition of TimeKeeper

// Class: TimeKeeper
// Derived from: OasisSystemObject
// Base for: OasisSystem
// Remarks: This class provides the functions to keep track of time, and convert to and from script object timestamps.

class TimeKeeper : public OasisSystemObject {
public:
    TimeKeeper(); // Constructor
    TimeKeeper(long Reference); // Constructor
    ~TimeKeeper(); // Destructor

    // Member functions for TimeKeeper
    long GetClockReference(); // Returns clock reference of timekeeper.
    void SetClockReference(long Reference); // Assigns clock reference of timekeeper.
    void StartSystemTime(); // Saves time of system start.
    void StartInputScriptTime(); // Saves time of start for input script.
    void StartOutputScriptTime(); // Saves time of start for output script.
    void SetSystemStartTime(long Seconds, long Microseconds); // Assigns system start time.
    void SetInputScriptStartTime(long Seconds, long Microseconds); // Assigns start time for input script file.
    void SetOutputScriptStartTime(long Seconds, long Microseconds); // Assigns start time for output script file.
    void CurrentSystemTimestamp(ScriptEvent& Event); // Assigns timestamp of script object to current system time (Absolute or Relative).
    void ScriptToSystemTimestamp(ScriptEvent& Event, long Reference); // Modifies timestamp of script object to reflect time reference to system time.
    void SystemToScriptTimestamp(ScriptEvent& Event, long Reference); // Assigns timestamp of script object to reflect reference to start of system time or start of input script file.
    void SetScriptDelay(ScriptEvent& Delay, long Reference); // Assigns time to disable script delay based on type of delay. Absolute delay is based on system start time. Relative is based on duration of delay.
    long ActiveScriptDelay(); // Returns TRUE if script delay is still active.
}
TimeKeeper* Instance();
    // Returns reference pointer of instance.
virtual char *ClassName();
    // Returns class identification string.

private:
    long ClockReference;
    // Time reference for time keeper.
TimeValue System_Starttime;
    // Start time of system.
TimeValue InputScript_Starttime;
    // Start time of input script.
TimeValue OutputScript_Starttime;
    // Start time of output script.
TimeValue End_Of_DelayTime;
    // Time to release script delay.
void ConvertToTimestamp(TimeValue *time, ScriptEvent &Event);
    // Converts system time to script object timestamp.
}; // end class TimeKeeper
APPENDIX F
Class Definition of OasisSystem

//----------------------------------------------------------------------
// Classname: OasisSystem
// Derived from: TimeKeeper, ScriptProcessor, ScriptGenerator
// Base for: OasisScriptSorter, OasisScriptPreprocessor
// Remarks: This class provides the mechanism for reading and writing script files. All script options are processed internally.
//----------------------------------------------------------------------
class OasisSystem : private TimeKeeper,
  private ScriptProcessor,
  private ScriptGenerator {

public:
  OasisSystem(); // constructor
  OasisSystem(long ClockReference); // constructor (use SYSTEM_CLOCK for reference in system time
  // access, or use USER_CLOCK for time provide by the user.
  ~OasisSystem(); // destructor

  // Member functions for OasisSystem
  long OpenInputScriptFile(char *Name); // Opens script file for input. Returns TRUE for successful file open.
  long OpenOutputScriptFile(char *Name); // Opens script file for output. Returns TRUE for successful file open.
  long GetScriptObject(ScriptObject& Script); // Requires ScriptObject as argument for script event I/O. The script object is returned with result of either - SCRIPT_ERROR, SCRIPT_VALID, END_OF_SCRIPT, SCRIPT_IN_DELAY, or NO_SCRIPT_EVENT.
  long PutScriptObject(ScriptObject& Script); // Writes script object to output script file and returns TRUE for successful file write.
  long CloseInputScriptFile(); // Closes input script file and returns result of TRUE for successful file close.
  long CloseOutputScriptFile(); // Closes input script file and returns result of TRUE for successful file close.
  double GetTimeFactor(); // Returns time factor for computing timestamps for all script objects.
  void SetTimeFactor(double Factor); // Assigns time factor for computing timestamps for all script objects.
void SetSystemStartTime(long Seconds, long Microseconds);
// Only used in USER_CLOCK time reference, the system
// startime is altered to match total seconds and
// microseconds. Start times for input and output script
// files will be the same for system start time.

void DisableScriptMessagesAndDelays();
// Sets flag for no script messages to be displayed and
// no script delays to be activated

OasisSystem* Instance();
// returns reference pointer of instance.

virtual char *ClassName();
// returns class identification string.

protected:
SourceScriptFile *CurrentInputFile;
// Reference to current input scriptfile

private:
long EndOfInputScript;
// Boolean flag for end of script file.
long DelayActive;
// Boolean flag for active script delay.
long MessagesAndDelays;
// Initially TRUE allowing script messages and delays
double TimeAdjustment;
// Amount of time to adjust script timestamps.
double TimeFactor;
// Time factor for each computed timestamp.
long TimeReference;
// Time reference for determining type of timestamp for
// script object. ABSOLUTE_TIME for assigning timestamps
// relative to system start time. RELATIVE_TIME for
// assigning timestamps relative to input script start
// time.

void ModifyScriptObject(ScriptEvent& Event);
// Modifies script object in computing timestamps according
// to time reference time adjustment, and time factor.

long ProcessScriptOption(ScriptOption& Option);
// Process script option for OASIS. Returns either
// SCRIPT_VALID or ENDOFSCRIPT.

long DelayScript(ScriptOption& Option);
// Assigns script delay based on type of reference in time
// delay. If absolute, delay is aborted when system time
// matches time of delay. If relative, time of delay is
// added to time of receiving script object for reference.

long ChainScript(ScriptOption& Option);
// Closes current script file and opens another. Returns
// null if error occurs.

long CallScript(ScriptOption& Option);
// Calls script file as a subroutine and returns back to
// calling script file. Returns null if error occurs
// during file open.
long RepeatScript(ScriptOption& Option);
    // Repeats current script file for number of iterations.
}; // end class OasisSystem
APPENDIX G
Class Definition of OasisScriptSorter

// ClassType: OasisScriptSorter
// Derived from: OasisSystem
// Base for: none
// Remarks: This class provides the mechanism to sort script
// objects of source script files and writes back to a destination script file. The qsort function of ANSI C is used for sorting.

class OasisScriptSorter : private OasisSystem {
public:
    OasisScriptSorter(); // Constructor
    ~OasisScriptSorter(); // Destructor

    // Member functions for OasisScriptSorter
    long PerformSort(char *Source, char *Destination);
    // Performs sort of source script file and generates sorted script to a destination script file. Source and Destination can be the same file name.
    OasisScriptSorter* Instance();
    // Returns reference pointer of instance.
    virtual char *ClassName();
    // Returns class identification string.

private:
    long ReadScriptFile(char *Source);
    // Reads script file into sort array and returns TRUE for valid script file or FALSE for invalid script file.
    long WriteScriptFile(char *Destination);
    // Writes script objects from sort array to destination script file. Returns TRUE if valid script write.
    long NumberOfEntries;
    // Returns number of script objects in sort array.
    ScriptSource *Script;
    // Array for sorting script objects.
    void CreateSortArray();
    // Creates sorting array.
    void DestroySortArray();
    // Destroys sorting array.
    double GetTimestamp(ScriptObject& Source);
    // Returns timestamp in total seconds and microseconds from script object.
    double PreviousTimestamp;
    // Timestamp of previous script object.
    long PreviousObjectWasAComment;
    // Boolean flag for previous script object.
}; // end OasisScriptSorter
APPENDIX H
Class Definition of OasisScriptPreprocessor

// Classtype: OasisScriptPreprocessor
// Derived from: OasisSystem
// Base for: none
// Remarks: This class provides a mechanism to check source script files errors prior to use.

class OasisScriptPreprocessor : private OasisSystem{
    public:
        OasisScriptPreprocessor(); // Constructor
        ~OasisScriptPreprocessor(); // Destructor
        // Member functions for OasisScriptPreprocessor
        long PerformErrorChecking(char *Source); // Reads script file and displays to standard output device of all errors in script file.
        OasisScriptPreprocessor* Instance(); // Returns reference pointer of instance.
        virtual char *ClassName(); // Returns class identification string.
}; // end OasisScriptPreprocessor
APPENDIX I

Class Definitions of Script Events and Attributes

// ==============================================================
// Class type: ObjectHeading
// Derived from: EventAttribute
// Base for: VehicleEvent, WeaponEvent, EnvironmentEvent
// Remarks: This class provides functions for object headings
// in degrees.
// ==============================================================
class ObjectHeading : public EventAttribute {
public:
    ObjectHeading(); // Constructor
    ~ObjectHeading(); // Destructor
    // Member functions for ObjectHeading
    void SetHeading(float Value); // Assigns object heading in degrees.
    float GetHeading(); // Returns object heading in degrees.
    void SetHeadingType(char Type); // Assigns heading type.
    char GetHeadingType(); // Returns heading type as a string (absolute or relative).
    ObjectHeading* Instance(); // Returns reference pointer of instance.
    virtual char* ClassName(); // Returns class identification string.
private:
    float Heading; // Heading of object in degrees.
    char HeadingType; // Heading in Relative direction 'r'
    // or Absolute direction 'a'
}; // end class ObjectHeading

// ==============================================================
// Class type: ObjectVelocity
// Derived from: EventAttribute
// Base for: VehicleEvent, WeaponEvent, EnvironmentEvent
// Remarks: This class provides functions for object velocity
// in kilometers per hour.
// ==============================================================
class ObjectVelocity : public EventAttribute {
public:
    ObjectVelocity(); // Constructor
    ~ObjectVelocity(); // Destructor

// Member functions for ObjectVelocity
float GetVelocity(); // Returns object velocity in km/h.
void SetVelocity(float Value); // Assigns object velocity in km/h.
ObjectVelocity* Instance(); // Returns reference pointer of instance.
virtual char *ClassName(); // Returns class identification string.

private:
float Velocity; // Velocity of object in kilometers per hour.

}; // end class ObjectVelocity;

// Class: ObjectPitchAngle
// Derived from: EventAttribute
// Base for: VehicleEvent, WeaponEvent
// Remarks: This class provides the functions for object pitch angle in degrees.
class ObjectPitchAngle : public EventAttribute {
public:
ObjectPitchAngle(); // Constructor
~ObjectPitchAngle(); // Destructor
// Member functions for ObjectPitchAngle
float GetPitchAngle(); // Returns object pitch angle in degrees.
void SetPitchAngle(float Angle); // Assigns object pitch angle in degrees.
virtual char *ClassName(); // Returns class identification string.

private:
float PitchAngle; // Object's pitch angle in degrees.

}; // end class ObjectPitchAngle;

// Class: ObjectRoll
// Derived from: EventAttribute
// Base for: VehicleEvent
// Remarks: This class provides functions for object roll.
class ObjectRoll : public EventAttribute {
public:
ObjectRoll(); // Constructor
~ObjectRoll(); // Destructor
// Member functions for ObjectRoll
float GetRollAngle(); // Returns roll angle in degrees.
void SetRollAngle(float Angle); // Assigns roll angle in degrees.
char *GetRollDirection(); // Returns roll direction of either "port" or "stbd"
void SetRollDirection(char *Direction); // Assigns roll direction.
ObjectRoll* Instance(); // Returns reference pointer of instance.
virtual char *ClassName(); // Returns class identification string.

private:
    float RollAngle; // Roll angle of object (in degrees)
    char *RollDirection; // Direction of pitch (port or stbd)
}; // end class ObjectRoll;

//-- Member functions for ObjectComponent
char *GetComponentName(); // Returns name of object component.
void SetComponentName(char *Name); // Assigns name to object component.
float GetComponentRotation(long Axis); // Returns component rotation in the requested axis.
void SetComponentRotation(float Xval, float Yval, float Zval); // Assigns component rotation in the X, Y, and Z axis.
float GetComponentTranslation(long Axis); // Returns component translation in the requested axis.
void SetComponentTranslation(float Xval, float Yval, float Zval); // Assigns component translation in the X, Y, and Z axis.
ObjectComponent* Instance(); // Returns reference pointer of instance.
virtual char *ClassName(); // Returns class identification string.

class ObjectComponent : public EventAttribute {
public:
    ObjectComponent(); // Constructor
    ~ObjectComponent(); // Destructor
    // Member functions for ObjectComponent
    char *GetComponentName(); // Returns name of object component.
    void SetComponentName(char *Name); // Assigns name to object component.
    float GetComponentRotation(long Axis); // Returns component rotation in the requested axis.
    void SetComponentRotation(float Xval, float Yval, float Zval); // Assigns component rotation in the X, Y, and Z axis.
    float GetComponentTranslation(long Axis); // Returns component translation in the requested axis.
    void SetComponentTranslation(float Xval, float Yval, float Zval); // Assigns component translation in the X, Y, and Z axis.
    ObjectComponent* Instance(); // Returns reference pointer of instance.
    virtual char *ClassName(); // Returns class identification string.

    // Class type: ObjectComponent
    // Derived from: EventAttribute
    // Base for: VehicleEvent
    // Remarks: This class provides functions for object component.
    // All movements of object component are represented
    // by the six degrees of freedom for translation and
    // rotation.

};
private:
char *ComponentName;
// Component name of object.
float ComponentRotation[XYZ];
// Degrees of freedom in rotation.
float ComponentTranslation[XYZ];
// Degrees of freedom in translation.
}; // end class ObjectComponent;

// Class type: ObjectWeapon
// Derived from: EventAttribute
// Base for: VehicleEvent
// Remarks: This class provides functions for object weapons,
// Usually components on vehicles.

class ObjectWeapon : public EventAttribute {
public:
    ObjectWeapon(); // Constructor
~ObjectWeapon(); // Destructor
    // Member functions for ObjectWeapon
    long GetWeaponRounds();
    // Returns number of rounds in weapon.
    void SetWeaponRounds(long Rounds);
    // Assigns number of rounds to weapon.
    char *GetWeaponName();
    // Return name of weapon.
    void SetWeaponName(char *Name);
    // Assigns name of weapon.
    ObjectWeapon* Instance();
    // Returns reference pointer of instance.
    virtual char *ClassName();
    // Returns class identification string.

private:
    char *WeaponName;
    // Weapon component name.
    long WeaponRounds;
    // Number of rounds in Weapon.
}; // end class ObjectWeapon
class ObjectExplosion : public EventAttribute {
public:
    ObjectExplosion();       // Constructor
    ~ObjectExplosion();      // Destructor
    // Member functions for ObjectExplosion
    char *GetExplosionDescription();
    // Returns description of explosion.
    void SetExplosionDescription(char *Description);
    // Assign description of object explosion.
    float GetExplosionBoundArea(long Axis);
    // Return coordinate of explosion bound area along requested axis.
    void SetExplosionBoundArea(float Xval, float Yval, float Zval);
    // Assign X, Y, and Z coordinates of explosion bound area.
    ObjectExplosion* Instance();
    // Returns reference pointer of instance.
    virtual char *ClassName();
    // Returns class identification string.

private:
    char *ExplosionDescription;
    // Descriptive type of explosion.
    float ExplosionBoundArea[XYZ];
    // Maximum area bound in X, Y, Z coordinates.
}; // end class ObjectExplosion;

class ObjectFireAndSmoke : public EventAttribute {
public:
    ObjectFireAndSmoke();   // Constructor
    ~ObjectFireAndSmoke();  // Destructor
    // Member functions for ObjectFireAndSmoke
    char *GetFireAndSmokeDescription();
    // Returns descriptive type of smoke or fire.
    void SetFireAndSmokeDescription(char *Description);
    // Assigns descriptive type of smoke or fire on object.
    float GetFireAndSmokeBoundArea(long Axis);
    // Returns coordinate of requested axis of maximum bound area of smoke or fire.

68
void SetFireAndSmokeBoundArea(float Xval, float Yval, float Zval);
// Assigns coordinates of X, Y, and Z axis maximum bound
// area of smoke or fire.
float GetFireAndSmokeOffset(long Axis);
// Returns coordinate of requested axis of smoke or fire
// offset on object.
void SetFireAndSmokeOffset(float Xval, float Yval, float Zval);
// Assigns coordinates of X, Y, and Z axis smoke or fire
// offset on object.
ObjectFireAndSmoke* Instance();
// Returns reference pointer of instance.
virtual char* ClassName();
// Returns class identification string.
private:
char* FireAndSmokeDescription;
// Descriptive type of smoke or flame on object.
float FireAndSmokeBoundArea[XYZ];
// maximum area bound in X, Y, and Z coordinates.
float FireAndSmokeOffset[XYZ];
// Offset coordinates on object.
}; // end class ObjectFireAndSmoke;

class ObjectTrail : public EventAttribute {
public:
 ObjectTrail(); // Constructor
 ~ObjectTrail(); // Destructor
 // Member functions for ObjectTrail
 char* GetTrailDescription();
// Returns description of object trail.
 void SetTrailDescription(char* Description);
// Assigns description to object trail.
 float GetTrailBoundArea(long Axis);
// Returns coordinate of trail bound area along request
// axis.
 void SetTrailBoundArea(float Xval, float Yval, float Zval);
// Assigns maximum bound area coordinates for X, Y, and Z.
 float GetTrailOffset(long Axis);
// Returns coordinate of trail offset along request axis.
 void SetTrailOffset(float Xval, float Yval, float Zval);
// Assigns object coordinates in X, Y, and Z for trail
// offset.
 ObjectTrail* Instance();
// Returns reference pointer of instance.

69
virtual char *ClassName(); // Returns class identification string.
private:
    char *TrailDescription; // Descriptive type of object trail.
    float TrailBoundArea[XYZ]; // Maximum area bound of trail.
    float TrailOffset[XYZ]; // Offset position on object.
}; // end class ObjectTrail;

//============================================================================================================
// Classtype: ObjectFuel
// Derived from: EventAttribute
// Base for: VehicleEvent
// Remarks: This class provides functions for object fuel.
//          Fuel is represented in Liters.
//============================================================================================================
class ObjectFuel : public EventAttribute {
public:
    ObjectFuel(); // Constructor
    ~ObjectFuel(); // Destructor
    float GetFuel(); // Returns amount of fuel in liters.
    void SetFuel(float Value); // Assigns amount of fuel in liters.
    virtual char *ClassName(); // Returns class identification string.
private:
    float Fuel; // Fuel state of Object (liters).
}; // end class ObjectFuel;

//============================================================================================================
// Classtype: EventObject
// Derived from: OasisEventObject
// Base for: VehicleEvent, WeaponEvent, MiscObjectEvent,
//          EnvironmentEvent
// Remarks: This class provides functions for all event objects
//============================================================================================================
class EventObject : public OasisEventObject {
public:
    EventObject(); // Constructor
    ~EventObject(); // Destructor
// Member functions for OasisEventObject
long ConvertIdFromLongInteger(long Id);
    // Converts a number to host and object identification
    // numbers.
long ConvertIdToLongInteger();
    // Returns number representing host and object id numbers.
long GetHostNumber();
    // Returns host number of object.
void SetHostNumber(long Number);
    // Assigns host number for object.
long GetObjectNumber();
    // Returns object identification number of object.
void SetObjectNumber(long Number);
    // Assigns object identification number to object.
char *GetObjectDescription();
    // Returns object description.
void SetObjectDescription(char *Description);
    // Assigns description to object.
char *GetObjectStatus();
    // Returns descriptive status of object.
void SetObjectStatus(char *Status);
    // Assigns descriptive status of object.
EventObject* Instance();
    // Returns reference pointer of instance.
virtual char *ClassName();
    // returns class identification string.

private:
    long ObjectHostNumber;
        // Host identification number of object.
    long ObjectNumber;
        // Object identification number.
    char *ObjectDescription;
        // Description type of object.
    char *ObjectStatus;
        // Descriptive status of object.
}; // end EventObject

(['/Classtype: WeaponSource
// Derived from: EventAttribute
// Base for: WeaponEvent
// Remarks: This class provides functions for weapon source.
//*******************************************************************************/
class WeaponSource : private EventAttribute {
    public:
    WeaponSource();
        // Constructor
-WeaponSource();
        // Destructor

71
// Member functions for WeaponSource
EventObject& GetWeaponSource();
    // Returns source object of weapon.
void SetWeaponSource(EventObject& Source);
    // Assigns source object of weapon.
WeaponSource* Instance();
    // Returns reference pointer of instance.
virtual char* ClassName();
    // Returns class identification string.

private:
    EventObject SourceOfWeapon;
    // Source object associated with weapon.
}; // end class WeaponSource

// Classtype:    TimeStamp
// Derived from: OasisEventObject
// Base for:     ScriptEvent
// Remarks:     This class provides functions for all timestamps.

class TimeStamp : public OasisEventObject {
public:
    TimeStamp();            // Constructor
    ~TimeStamp();           // Destructor
    // Member functions for TimeStamp
    void SetHours(long Value);
        // Assigns hours in timestamp.
    long GetHours();
        // Returns hours in timestamp.
    void SetMinutes(long Value);
        // Assigns minutes in timestamp.
    long GetMinutes();
        // Returns minutes in timestamp.
    void SetSeconds(long Value);
        // Assigns seconds in timestamp.
    long GetSeconds();
        // Returns seconds in timestamp.
    void SetMicroseconds(long Value);
        // Assigns microseconds in timestamp.
    long GetMicroseconds();
        // Returns microseconds in timestamp.
    long StringToTimestamp(char* Timestring);
        // Converts timestamp string to hours, minutes, seconds,
        // and microseconds.
    char* TimestampToString();
        // Returns timestamp as a string.
    TimeStamp* Instance();
        // Returns reference pointer of instance.
    virtual char* ClassName();
        // Returns class identification string.

72
private:
   long Hours;
   // Timestamp hours.
   long Minutes;
   // Timestamp minutes.
   long Seconds;
   // Timestamp seconds.
   long Microseconds;
   // Timestamp microseconds.
   char *TimeString;
   // Timestamp in string format.
}; // end class TimeStamp

class EventPosition : public OasisEventObject {
public:
   EventPosition(); // Constructor
   ~EventPosition(); // Destructor
   // Member functions for EventPosition
   void SetGridzone(char *NewGridzone);
   // Assigns gridzone of UTM position
   char *GetGridzone();
   // Returns gridzone of UTM coordinates.
   void SetMgrs(char *NewMgrs);
   // Assigns MGRS of UTM coordinates.
   char *GetMgrs();
   // Returns MGRS of UTM coordinates.
   void SetNorthing(long Value);
   // Assigns Northing of UTM coordinates.
   long GetNorthing();
   // Returns Northing of UTM coordinates.
   void SetEasting(long Value);
   // Assigns Easting of UTM coordinates.
   long GetEasting();
   // Returns Easting of UTM coordinates.
   void SetElevation(float Value);
   // Assigns elevation.
   float GetElevation();
   // Returns elevation.
   long StringToUtmPosition(char *Position, char *defaultGZ, char *defaultMGRE);
   // Extracts UTM coordinates from Position, and returns null
   // if error occurs. Default gridzone and mgrs are required
   // for UTM defaults.
long StringToMap(char *map_strg, char *defaultGZ,
    char *defaultMGRS);
    // Extracts default UTM gridzone and/or Mgrs, and returns
    // null if error occurs.
char *UtmPositionToString();
    // Returns UTM coordinates as a string.
EventPosition* Instance();
    // Returns reference pointer of instance.
virtual char *ClassName();
    // Returns class identification string.
private:
    char *Gridzone;
        // grid zone of UTM coordinate system
    char *Mgrs;
        // mgrs of UTM coordinate system
    long Northing;
        // northing offset of grid zone
    long Easting;
        // easting offset of grid zone
    float Elevation;
        // altitude
    char *PositionString;
        // UTM coordinates in string format
}; // end class EventPosition;

class ScriptEvent : public EventObject, public EventPosition,
                    private EventAttribute, public TimeStamp {
public:
    ScriptEvent(); // Constructor
    ~ScriptEvent(); // Destructor
    // Member functions for ScriptEvent
    long GetEventType();
        // Returns type of script event.
    void SetEventType(long Type);
        // Assigns type of script event.
    char *GetEventResult();
        // Returns descriptive result of event.
    void SetEventResult(char *Result);
        // Assigns descriptive result of event.


ScriptEvent* Instance();
// Returns reference pointer to instance.
virtual char* ClassName();
// Returns class identification string.
private:
    long EventType;
// Type of script event.
    char* EventResult;
// Description result of event.
}; // end class ScriptEvent

// Classtype: VehicleEvent
// Derived from: ScriptEvent, ObjectHeading, ObjectVelocity,
// ObjectPitchAngle, ObjectRoll, ObjectComponent,
// ObjectWeapon, ObjectExplosion, ObjectFireAndSmoke,
// ObjectTrail, ObjectFuel
// Base for: ScriptOL::ct
// Remarks: This class is for all vehicle events.
class VehicleEvent : public ScriptEvent, public ObjectHeading,
    public ObjectVelocity, public ObjectPitchAngle,
    public ObjectRoll, public ObjectComponent,
    public ObjectWeapon, public ObjectFireAndSmoke,
    public ObjectTrail, public ObjectExplosion {
public:
    VehicleEvent(); // Constructor
    ~VehicleEvent(); // Destructor
    // Member functions for VehicleEvent
    VehicleEvent*Instance();
    // Returns reference pointer of instance
    virtual char* ClassName();
    // Returns class identification string.
}; // end class VehicleEvent

// Classtype: WeaponEvent
// Derived from: ScriptEvent, WeaponSource, ObjectHeading,
// ObjectPitchAngle,
// ObjectVelocity, ObjectExplosion
// Base for: ScriptObject
// Remarks: This class is for all Weapon events.
class WeaponEvent : public ScriptEvent, public WeaponSource,
    public ObjectHeading, public ObjectPitchAngle,
    public ObjectVelocity, public ObjectExplosion {
public:
    WeaponEvent();           // Constructor
    ~WeaponEvent();          // Destructor
    // Member functions for WeaponEvent
    WeaponEvent* Instance();
        // Returns reference pointer of instance.
    virtual char *ClassName();
        // Returns class identification string.
}; // end class WeaponEvent

reamble

class MiscObjectEvent : public ScriptEvent, public ObjectFireAndSmoke, public ObjectExplosion
{
public:
    MiscObjectEvent();       // Constructor
    ~MiscObjectEvent();      // Destructor
    // Member functions for MiscObjectEvent
    MiscObjectEvent* Instance();
        // Returns reference pointer of instance.
    virtual char *ClassName();
        // Returns class identification string.
}; // end class MiscObjectEvent

class EnvironmentEvent
{
public:
    EnvironmentEvent();        // Constructor
    ~EnvironmentEvent();       // Destructor
    // Member functions for EnvironmentEvent
    EnvironmentEvent* Instance();
        // Returns reference pointer of instance.
    virtual char *ClassName();
        // Returns class identification string.
}; // end class EnvironmentEvent
class ScriptComment : public ScriptEvent {
    // Data structure for each comment line
    typedef struct CommentPointer {
        char *Line;       // Pointer to comment line.
        struct CommentPointer *Next; // Pointer to next comment line.
    } CommentLink;

    public:
    ScriptComment();          // Constructor
    ~ScriptComment();         // Destructor

    // Member functions for ScriptComment
    long ReadFromFile(FILE *InputFile, char *Line){
        // Reads from script file rest of comment block. Prior to
        // function call, first script line is already read.
        void DeleteComment();       // Removes all comment lines in comment block.
    long GetNumberOfCommentLines();     // Returns number of lines in comment block.
    CommentLink *GetCommentBlock();      // Returns reference pointer to comment block.
    void SetCommentBlock(CommentLink *Block);  // Assigns reference pointer to comment block.
    ScriptComment* Instance();            // Returns reference pointer of instance.
    virtual char *ClassName();            // Returns class identification string.

    private:
    long NumberOfCommentLines;   // Number of lines in comment block.
    CommentLink *Comment;        // Reference pointer to next comment line.
}; // end class ScriptComment

class ScriptOption : public ScriptEvent {
    public:
    ScriptOption();          // Constructor
    ~ScriptOption();         // Destructor
// Member functions for ScriptOption
double GetScriptTimeFactor();
// Returns script time factor.
void SetScriptTimeFactor(double Factor);
// Assigns script time factor.
double GetScriptTimeAdjustment();
// Returns script time adjustment.
void SetScriptTimeAdjustment(double Adjustment);
// Assigns script time adjustment.
char *GetDelayType();
// Returns script delay type.
void SetDelayType(char *Type);
// Assigns script delay type.
char *GetTimeReference();
// Returns script time reference.
void SetTimeReference(char *Reference);
// Assigns time reference for assigning script timestamps.
char *GetScriptLine();
// Returns script line or message.
void SetScriptLine(char *Line);
// Assigns script line or message.
char *GetScriptFilename();
// Returns filename of script file.
void SetScriptFilename(char *Name);
// Assigns filename of script file.
long GetScriptRepeatValue();
// Returns script repeat value of iterations.
void SetScriptRepeatValue(long Value);
// Assigns script repeat value of iterations.
long GetScriptObjectNumberIncrement();
// Returns script object no. increment.
void SetScriptObjectNumberIncrement(long Value);
// Assigns script object number increment.
ScriptOption* Instance();
// Returns reference pointer of instance.
virtual char *ClassName();
// Returns class identification string.

private:
char *DelayType;
// Delay type in relative or absolute.
char *TimeReference;
// Time reference in relative or absolute.
char *ScriptLine;
// Reference pointer for script line or message.
char *ScriptFilename;
// Filename of script file.
long RepeatValue;
// Number of repeats for script file.
long ObjectNumberIncrement;
// Object number increment for each iteration.
double TimeFactor;
    // Time factor for computing timestamps.
double TimeAdjustment;
    // Time adjustment for computing timestamps.
}; // end class ScriptOption
APPENDIX J
Class Definition of ScriptObject

//----------------------------------------------------------------------------------
// Class type: ScriptObject
// Derived from: VehicleEvent, WeaponEvent, MiscObjectEvent,
//               EnvironmentEvent, ScriptOption, ScriptComment
// Base for: none
// Remarks: This class provides functions for accessing script events.
//----------------------------------------------------------------------------------
class ScriptObject : public VehicleEvent,
                   public WeaponEvent,
                   public MiscObjectEvent,
                   public EnvironmentEvent,
                   public ScriptOption,
                   public ScriptComment {

public:
  ScriptObject();  // Constructor
  ~ScriptObject(); // Destructor

  // Member functions for ScriptObject
  long GetScriptEventType();  // Returns type of script event.
  void SetScriptEventType(long Type);  // Assigns type of script event.
  MiscObjectEvent& GetMiscObjectEvent();  // Returns reference of misc object event.
  ScriptOption& GetScriptOption();  // Returns reference of script option.
  ScriptComment& GetScriptComment();  // Returns reference of script comment.
  virtual char *ClassName();  // Returns class identification string.

private:
  long ScriptEventType; // Type of event for script object.
}; // end class ScriptObject
APPENDIX K
NPSNET-OASIS Network Interface

The following NPSNET data structure is used for sending vehicle updates over the net. Attributes marked with an asterisk are used in the vehicle update message packet.

```c
struct vehpostype {
    int vehtype;               *
    int control;               *
    int gunfire;               *
    int alive;                 *
    int rounds;
    int deadframes;
    int coll_interval;
    float pos[3],             *
       eye[3],
       lookatpt[3],
       lookfrompt[3];
    float direction,           *
       viewdirection,         *
       elev,                 *
       gunelev,             *
       speed,               *
       roll,                *
       pitch,              *
       gas,                 *
       coll_range;
};// end structure
```

The following is an example of the transformation of a VEHICLE_ACTIVATE script event to a data structure in the local state of the world in NPSNET-OASIS network interface.

```c
Vehicle[Number].vehtype = LookForObjectTypeNumber(
    ScriptEvent.GetObjectDescription());
Vehicle[Number].control = SCRIPTED;
Vehicle[Number].gunfire = 0;
Vehicle[Number].alive = TRUE;
Vehicle[Number].rounds = DEFAULT_VEHICLE_ROUNDS;
Vehicle[Number].deadframes = 0;
Vehicle[Number].coll_interval = 0;
Vehicle[Number].pos[X] = ScriptEvent.GetEasting();
Vehicle[Number].pos[Z] = MAX_UTM_NORTHING -
    ScriptEvent.GetNorthing();
Vehicle[Number].elev = ScriptEvent.GetElevation();
Vehicle[Number].eye[X] = 0.0;
```
Vehicle[Number].eye[Y]  = 0.0;
Vehicle[Number].eye[Z]  = 0.0;
Vehicle[Number].lookatpt[X]  = 0.0;
Vehicle[Number].lookatpt[Y]  = 0.0;
Vehicle[Number].lookatpt[Z]  = 0.0;
Vehicle[Number].lookfrompt[X]  = 0.0;
Vehicle[Number].lookfrompt[Y]  = 0.0;
Vehicle[Number].lookfrompt[Z]  = 0.0;
Vehicle[Number].direction  = ScriptEvent.GetHeading();
Vehicle[Number].viewdirection  = 0.0;
Vehicle[Number].elev  = ScriptEvent.GetElevation();
Vehicle[Number].gunelev  = 0.0;
Vehicle[Number].speed  = ScriptEvent.GetVelocity();
Vehicle[Number].roll  = 0.0;
Vehicle[Number].pitch  = 0.0;
Vehicle[Number].gas  = DEFAULT_VEHICLE_GAS;
Vehicle[Number].coll_range  = 0.0;
LIST OF REFERENCES


INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Dudley Knox Library 2
   Code 52
   Naval Postgraduate School
   Monterey, California 93943-5100

3. Dr. Michael J. Zyda 8
   Code CS/Zk, Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93943-5100

4. David R. Pratt 2
   Code CS/Pr, Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93943-5100

5. Lt. Phillip D. West, USN 1
   898 Rock Street
   Archbald, Pennsylvania 18403