THESIS

ELECTRONIC MANEUVERING BOARD AND DEAD RECKONING TRACER DECISION AID FOR THE OFFICER OF THE DECK

by

Joey L. Frantzen
Kenneth L. Ehresman

September 2001

Thesis Co-Advisors: Richard D. Riehle Luqi

Approved for public release; distribution is unlimited.
## Report Documentation Page

<table>
<thead>
<tr>
<th>Report Date</th>
<th>Report Type</th>
<th>Dates Covered (from... to)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 Sep 2001</td>
<td>N/A</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title and Subtitle</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Electronic Maneuvering Board and Dead Reckoning Tracer Decision Aid for the Officer of the Deck</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contract Number</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Grant Number</td>
<td></td>
</tr>
<tr>
<td>Program Element Number</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Author(s)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Joey L. Frantzen &amp; Kenneth L. Ehresman</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project Number</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Number</td>
<td></td>
</tr>
<tr>
<td>Work Unit Number</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performing Organization Name(s) and Address(es)</th>
<th>Performing Organization Report Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research Office Naval Postgraduate School Monterey, Ca 93943-5138</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sponsoring/Monitoring Agency Name(s) and Address(es)</th>
<th>Sponsor/Monitor's Acronym(s)</th>
<th>Sponsor/Monitor's Report Number(s)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Distribution/Availability Statement</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Approved for public release, distribution unlimited</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supplementary Notes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abstract</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject Terms</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Report Classification</th>
<th>Classification of this page</th>
</tr>
</thead>
<tbody>
<tr>
<td>unclassified</td>
<td>unclassified</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classification of Abstract</th>
<th>Limitation of Abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>unclassified</td>
<td>UU</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of Pages</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>210</td>
<td></td>
</tr>
</tbody>
</table>
REPORT DOCUMENTATION PAGE

Electronic Maneuvering Board and Dead Reckoning Tracer Decision Aid for the Officer of the Deck

Joey L. Frantzen & Kenneth L. Ehresman

Naval Postgraduate School
Monterey, CA 93943-5000

The purpose of this research is to implement a stand-alone system that will provide timely and accurate contact information for Decision-Makers. By creating a reliable, automated system in a format that is familiar to all Surface Warfare Officers we will provide the Navy with a valuable decision-making tool, while increasing ease of data exchange and reducing current redundancies and manning inefficient practices.

Our software design is based upon the Unified Modeling Language (UML). UML allows us to construct a software model that is supported by the Ada programming language. Our design is based upon these fundamental tenants: Non-Operating System dependent, Non-Hardware System dependent, Extensible and Modular design. Ada provides a certified compiler, making our code robust and assuring the “buyer” that the program does what we advertise it to do.

The U.S. Navy currently bases the majority of our contact management decisions around a time and manning intensive paper-based Maneuvering Board process. Additional manning requirements are involved on many Naval Ships in order to accurately convey the information to the OOD and/or the Commanding Officer. When given situations where there exist multiple contacts, the current system is quickly overwhelmed and may not provide Decision-Makers a complete and accurate picture in a timely manner.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

The U.S. Navy currently bases the majority of our contact management decisions around a time and manning intensive paper-based Maneuvering Board process. Additional manning requirements are involved on many Naval Ships in order to accurately convey the information to the OOD and/or the Commanding Officer. When given situations where there exist multiple contacts, the current system is quickly overwhelmed and may not provide Decision-Makers a complete and accurate picture in a timely manner.

The purpose of this research is to implement a stand-alone system that will provide timely and accurate contact information for Decision-Makers. By creating a reliable, automated system in a format that is familiar to all Surface Warfare Officers we will provide the Navy with a valuable decision-making tool, while increasing ease of data exchange and reducing current redundancies and manning inefficient practices.

Our software design is based upon the Unified Modeling Language (UML). UML allows us to construct a software model that is supported by the Ada programming language. Our design is based upon these fundamental tenants: Non-Operating System dependent, Non-Hardware System dependent, Extensible and Modular design. Ada provides a certified compiler, making our code robust and assuring the “buyer” that the program does what we advertise it to do.


15. NUMBER OF PAGES: 197

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified

20. LIMITATION OF ABSTRACT UL
Approved for public release; distribution is unlimited.

ELECTRONIC MANEUVERING BOARD AND DEAD RECKONING TRACER
DECISION AID FOR THE OFFICER OF THE DECK

Joey L. Frantzen
Lieutenant, United States Navy
B.S., United States Naval Academy, 1994

Kenneth L. Ehresman
Lieutenant, United States Navy
B.S., University of Maryland, 1995

Submitted in partial fulfillment of the requirements for the degree of

MASTHER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2001

Authors:

Joey L. Frantzen
Kenneth L. Ehresman

Approved by:

Richard D. Riehle; Thesis Co-Advisor

Dr. Luqi, Thesis Co-Advisor

Christopher S. Eagle, Chairman
Computer Science Department
ABSTRACT

The U.S. Navy currently bases the majority of our contact management decisions around a time and manning intensive paper-based Maneuvering Board process. Additional manning requirements are involved on many Naval Ships in order to accurately convey the information to the OOD and/or the Commanding Officer. When given situations where there exist multiple contacts, the current system is quickly overwhelmed and may not provide Decision-Makers a complete and accurate picture in a timely manner.

The purpose of this research is to implement a stand-alone system that will provide timely and accurate contact information for Decision-Makers. By creating a reliable, automated system in a format that is familiar to all Surface Warfare Officers we will provide the Navy with a valuable decision-making tool, while increasing ease of data exchange and reducing current redundancies and manning inefficient practices.

Our software design is based upon the Unified Modeling Language (UML). UML allows us to construct a software model that is supported by the Ada programming language. Our design is based upon these fundamental tenants: Non-Operating System dependent, Non-Hardware System dependent, Extensible and Modular design. Ada provides a certified compiler, making our code robust and assuring the “buyer” that the program does what we advertise it to do.
# TABLE OF CONTENTS

I. INTRODUCTION ........................................................................................................1  
   A. PROBLEM STATEMENT .............................................................................1

II. BACKGROUND ..........................................................................................................3  
    A. TRADITIONAL DECISION-MAKING PROCESS ....................................3  
       1. The OODA Loop ...............................................................................3  
           a. Observation ................................................................................4  
           b. Orientation ..............................................................................4  
           c. Decision ....................................................................................4  
           d. Action .......................................................................................4  
    B. INCREASING COLLISIONS AT SEA .........................................................5  
    C. PAPER VS. DIGITAL ..............................................................................7  
    D. MANNING THE FUTURE NAVY .............................................................9

III. OVERALL SOFTWARE DESIGN ..........................................................................11  
    A. UNIFIED MODELING LANGUAGE ......................................................11  
       1. Use-Case Diagram ........................................................................11  
       2. Class Diagram ...............................................................................12  
           a. DRT Class ..............................................................................12  
           b. Date Class .............................................................................13  
           c. Time Class ............................................................................13  
           d. Latitude/Longitude Class ........................................................14  
           e. Hit Class ...............................................................................15  
           f. Track Class ............................................................................16  
           g. OwnShip Class .................................................................17  
           h. GPS Class .............................................................................18  
           i. Speed Class ...........................................................................18  
           j. Realnum Class ........................................................................19  
           k. Degree Class ...........................................................................19  
           l. Radar Class ............................................................................19  
           m. Network Class .......................................................................19  
           n. CPA Class ..............................................................................20  
           o. Moboard Class .........................................................................20  
           p. MainScreen-Pkg Class ............................................................21  
           q. MainScreen-Pkg-Callbacks Class ..........................................22  
           r. Sketchpad Class .................................................................23  
           s. Utilities Class ...........................................................................24  
           t. File IO Class ............................................................................24  
           u. Historical IO Class .................................................................25  
           v. Wind Class ...............................................................................26  
           w. GtkAda and GNAT ....................................................................27  
       3. Deployment Diagram ...........................................................................26  
       4. GtkAda and GNAT ...........................................................................27
LIST OF FIGURES

Figure 1. The OODA Loop for Officer of the Deck (From: OOD) .............................................4
Figure 2. The OODA Loop Cycle ..........................................................................................5
Figure 3. USS DENVER (LPD 9) pulls into Pearl Harbor Following An at-Sea Collision .......7
Figure 4. The Paper-based Moboard.................................................................7
Figure 5. The Digital Moboard ..................................................................................8
Figure 6. Picture of DD-21 Concept ........................................................................9
Figure 7. Use-Case Diagram .........................................................................................12
Figure 8. Dates Class .................................................................................................13
Figure 9. Times Class ...............................................................................................14
Figure 10. Lat_Long Class .........................................................................................15
Figure 11. Hit Class ..................................................................................................16
Figure 12. Track Class .............................................................................................17
Figure 13. Ownship Class .........................................................................................18
Figure 14. Speed Class ..............................................................................................19
Figure 15. Realnum Class .........................................................................................19
Figure 16. Degree Class ............................................................................................19
Figure 17. CPA Class ..............................................................................................20
Figure 18. Moboard Class .........................................................................................21
Figure 19. MainScreen-Pkg Class .............................................................................22
Figure 20. MainScreen-Pkg-Callbacks Class ..............................................................23
Figure 21. Sketchpad Class ......................................................................................23
Figure 22. Utilities Class .........................................................................................24
Figure 23. File_IO Class ..........................................................................................25
Figure 24. Historical IO Class ..................................................................................25
Figure 25. Wind Class ..............................................................................................26
Figure 26. Deployment Diagram ..............................................................................26
Figure 27. GtkAda layered structure .........................................................................27
Figure 28. Screenshot of GVD Version 1.2.1 ...............................................................29
THIS PAGE INTENTIONALLY LEFT BLANK
ACKNOWLEDGMENTS

Sincere appreciation to

Dr. Luqi & Mr. Richard Riehle

for their guidance and support in this endeavor.
I. INTRODUCTION

A. PROBLEM STATEMENT

The U.S. Navy currently bases the majority of our contact management decisions around a time and manning intensive paper-based Maneuvering Board process. The use of Maneuvering Boards is a perishable skill that has a steep learning curve. In order to overcome inherent human error, it is not uncommon to have up to four people simultaneously involved in solving just one maneuvering problem. Additional manning requirements are involved on many Naval Ships in order to accurately convey the information to the Officer of the Deck (OOD) and/or the Commanding Officer. When given situations where there exist multiple contacts, the current system is quickly overwhelmed and may not provide Commanding Officers and OODs a complete and accurate picture in a timely manner.

Since 1996, there has been an increase in the number of collisions at sea, resulting in the loss of millions of dollars and thousands of operational hours for ships that are critical to our force structure. Although the time-tested method we use to make maneuvering decisions works, its technology has not kept pace with the increase in the ocean’s traffic density. What is required is a faster and more accurate means by which this method is executed.

The purpose of this research is to implement a stand-alone system that will provide timely and accurate contact information for U.S. Navy Commanding Officers, OODs, and CIC watch teams. By creating a reliable, automated system in a format that is familiar to all Surface Warfare Officers we will provide the Navy with a valuable decision-making tool, while increasing ease of data exchange and reducing current redundancies and manning inefficient practices.

Our software design is based upon the Unified Modeling Language (UML). UML allows us to construct a software model that is supported by the Ada programming language. UML also provides significant benefits to us, as software engineers, by helping to build rigorous, traceable and maintainable models that will support the software development cycle. Our design is based upon these fundamental tenants: Non-
Operating System dependent, Non-Hardware System dependent, Extensible and Modular design. Ada provides a certified compiler and environment, making our code robust and assuring the “buyer” that the program does what we advertise it to do. We also chose Ada because of the Re-usability inherent to the modular design structure. Our program does not use hardware specific libraries/architecture such as MFC.

This system will significantly enhance Safe Navigation at Sea while maintaining the age old, time tested ways of avoiding other vessels at sea.

Our software design is implemented via using Gtk-Ada, which allows the development of a GUI-based program that is neither operating system nor hardware dependent. Gtk-Ada is supported on a wide range of platforms, and its use can ultimately allow the U.S Navy to develop operational tools and programs without being limited to any specific hardware or operating system. The flexibility that this tool affords can reduce development and maintenance cost significantly as the amount of rework due to a paradigm shift in any operating system vendor will be greatly reduced or eliminated. Additionally, Ada’s ability to interoperate with other programming languages make it an excellent candidate for integrating with current stove-piped systems.

Our system provides the basis for a robust fusion analysis plot that, due to its modular design, can interact with virtually any other system.
II. BACKGROUND

A. TRADITIONAL DECISION-MAKING PROCESS

Prior to Maneuvering Boards, the traditional mariner relied upon the seaman’s eye and the knowledge gained from many hours of standing watches on the bridge. This knowledge pool helped the ship driver make the right decision when confronted with other vessels. The evolution of radar allowed vessels to see contacts at great distances and measure the bearing and ranges of those contacts. The Maneuvering Board quickly followed the radar allowing ship drivers an alternate visual representation of radar contacts based upon trigonometric fundamentals. This now allowed OODs and Commanding Officers a better way to frame the problem in more concrete terms.

1. The OODA Loop

The OOD decision-making process is designed to try and reduce uncertainty by gathering information, and transforming this information into knowledge and understanding. The utilization of radars and Maneuvering Boards aids a Commander/OOD in reducing the level of uncertainty. This decision process is known as the OODA Loop: Observation, Orientation, Decision, and Action.

Whenever trying to establish Command and Control there exists two fundamental factors that shape the environment: uncertainty and time. The Moboard model lies within the Orientation phase of the OODA Loop. The Electronic Maneuvering Board Decision Aid reduces the level of uncertainty and the amount of time inherent to the Maneuvering Board process.

- **Observation**
  - Naked Eye
  - Binoculars (Lookouts)
  - Surface radar
  - Sonar
  - Audible noise
  - Compass (Magnetic/Gyroscope)
- **Orientation**
  - Moboards
  - Dead Reckoning Trace
  - Experience
  - Assumptions
  - Estimates
  - Judgments
• Bearings
• Altimeter

• Issue orders

* Decide course of action

Figure 1. The OODA Loop for Officer of the Deck (From: OOD)

a. Observation

Observes the environment (using all sensors, information systems, and situational reports from his subordinates) to collect data about their surroundings and the status of contacts. This data is typically correlated, fused, and displayed in a common tactical picture—a representation or image of the contact space. A Commander or OOD had several methods of retrieving data via visual lookouts, surface radars, sonar, and/or his/her own eyes.

b. Orientation

A Commander/OOD orients themselves to the environment—that is, they form a mental picture of the situation—by converting sensor data and other information into estimates, assumptions, and judgments about what is happening. From this orientation a commander/OOD derives his understanding of the contact space, or situational awareness.

c. Decision

Based on the understanding derived from his/her Orientation, the commander/OOD then decides on a course of action and comes up with a plan.

d. Action

The commander/OOD sets forth his intent and issues orders to put that plan into action.
B. INCREASING COLLISIONS AT SEA

Since 1996, there has been a marked increase in the number of collisions at sea, resulting in the loss of millions of dollars and thousands of operational hours for ships that are critical to our force structure. A Navy investigation into the collision of USS
Denver (LPD 9) with USNS Yukon (T-AO 202) found that Captain should have realized his ship was on a collision course with the oiler. In hindsight, had the CO of the Denver had more time to make his critical maneuvering decision and had he been given more accurate contact information in a timelier manner, the CO of the Denver would never have made such a critical mistake.

There are many variables that play a significant part in the reasons for more frequent collisions at sea over the past 5 years. These factors range from inexperience, training, crew fatigue, Op Tempo, and higher traffic densities on today’s seas. The end result is OODs and CO’s who may not have complete situational awareness, who become complacent, and decision-makers who don’t receive safety critical information in a timely and accurate manner. The primary issue is not the decisions that are made when it comes to maneuvering, but the information that decision-makers have when making those decisions. Although collisions are a high profile issue, it’s the numerous and countless “near misses” that go unreported and often untreated. Looking back into our crystal ball we can see many instances where Commanding Officers and OODs could have benefited from a better system and a better means by which contact information was being displayed and presented to them. The time-tested method we use to make maneuvering decisions works. The problem is that technology has not kept pace with the increase in the ocean’s traffic density. What is required is a faster and more accurate means by which this method is executed.
C. PAPER VS. DIGITAL

Figure 3. USS DENVER (LPD 9) pulls into Pearl Harbor Following An at-Sea Collision

Figure 4. The Paper-based Moboard
The argument for or against traditional paper-based Moboards versus Digital-Based Moboards is based upon two simple factors.

Traditional paper-based Moboards are done with a pencil and straightedge. This process can be inaccurate and is often prone to human error. Even a very experienced sailor can make mistakes when doing a Moboard solution, especially in time critical situations, periods of rough seas, night time operations, or situations where there are multiple contacts.

Digital-based Moboards will speed this process up and eliminate the inherent human error innate to the paper-based Moboards process. By decreasing the time required to produce a Moboard solution it in turn decreases the time required to complete the orientation process and thus speeds up the overall decision process. Having more time and more accurate information in an understandable and easy to assimilate presentation is every Commander’s desire. This is what Digital-based Moboards provide.

Figure 5. The Digital Moboard
D. MANNING THE FUTURE NAVY

Figure 6. Picture of DD-21 Concept

With the evolution of “Smart Ship” and the DD-21 initiative the manning of Navy Ships has become a high profile issue. The future Navy will no longer have the luxury of 350-manned combatant ships. The Navy of the future will require less men and women who are more technically proficient and better trained. The bridge of the next generation will still rely on good seamanship, experience, and a trained eye while instead of using paper-based tools, the tasks and aids used to process contacts and information will be done in a digital-based medium. The modern Navy will have to depend on exceptional sensors and computer systems that are able to frame an abundance of information into a manageable and clear presentation. The Electronic Maneuvering Board Decision Aid is designed to meet this emerging need. With this computerized decision tool the requirements for multiple junior officers doing Mobords or several Operation Specialists in Combat maintaining a DRT contact picture will be reduced greatly. GPS will automatically be updated into the system, instantly giving the Commanding Officer and OOD Latitude and Longitude information of all the local area contacts at the mere click of the mouse.

Additionally our computer program will have the ability to maintain a digital log, vice having a paper-based Deck Log maintained by the Quartermaster (QM). This may be another avenue by which the U.S. Navy can reduce the manning requirements on the bridge while maintaining and improving upon the safety of ships at sea. The Quartermaster will no longer be required to log each course and speed change, OOD watch changes, casualties, etc. This will all be maintained in a central database allowing for a visual playback of events for any post-operations analysis. This feature will allow
the evaluator to view a list of events as well as display a visual contact picture chronologically corresponding with these logged events. Thus, the end result is better post-operations analysis and understanding of the environment on the bridge at the time of the operation, mishap, or exercise.
III. OVERALL SOFTWARE DESIGN

A. UNIFIED MODELING LANGUAGE

Our software design is based upon the Unified Modeling Language (UML). UML allows us to construct a software model that is supported by the Ada programming language. UML also provides significant benefits to us, as software engineers, by helping to build rigorous, traceable and maintainable models that will support the software development cycle. Our design is based upon these fundamental tenants: Non-Operating System dependent, Non-Hardware System dependent, Extensible and Modular design. Ada provides a certified compiler and environment, making our code robust and assuring the “buyer” that the program does what we advertise it to do. We also chose Ada because of the Re-usability inherent to the modular design structure. Our program does not use hardware specific libraries/architecture such as MFC.

Our design was based upon the following UML diagrams: Use-Case, Sequence, Class, Object, and Deployment Diagrams.

1. Use-Case Diagram

The Use-Case Diagram was based upon what an OOD or CO would require the computer system to do. Basic functions such as Plot Contact, Display Contact, Calculate Contact Course and Speed, and Calculate CPA were the primary Use-cases for the basic computer program. Other actors such as Global Positioning System and Generic Radar System will be required for implementation at a later date. All of these actors interact with two systems, Dead Reckoning Trace and Maneuvering Board. These two systems are just separate views of the same data, framing our visual representation for the user.
2. **Class Diagram**

To truly make a computer program reusable, modular, and maintainable the programmer must adhere to a strict Objected Oriented Methodology (OOM). Understanding this key concept, the programmer must build his/her classes in an Object Oriented approach. With this in mind, our Electronic Moboard and DRT was structured to be modular, maintainable, and reusable.

**a. DRT Class**

*Description:* The DRT Class is similar to the Moboard Class in that it displays information from the Ownship and Tracks Class. The main difference is the DRT Class presents a true picture vice a relative picture presented in the Moboard Class.


b. Date Class

Description: The Date Class contains a type Date (Day, Month, Year).

The Class includes Functions and Procedures to Get a Date from the computer system or GPS, as well as, return all the Date values and set all the Date values.

<table>
<thead>
<tr>
<th>dates Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date : Record</td>
</tr>
<tr>
<td>Month : Month_Number = Calls Ada.Calendar</td>
</tr>
<tr>
<td>Day : Day_Number = Calls Ada.Calendar</td>
</tr>
<tr>
<td>Year : Year_Number = Calls Ada.Calendar</td>
</tr>
<tr>
<td>Get_Date_From_System()</td>
</tr>
<tr>
<td>Set_Date()</td>
</tr>
<tr>
<td>Set_Day()</td>
</tr>
<tr>
<td>Set_Month()</td>
</tr>
<tr>
<td>Set_Year()</td>
</tr>
<tr>
<td>Get_Day()</td>
</tr>
<tr>
<td>Get_Month()</td>
</tr>
<tr>
<td>Get_Year()</td>
</tr>
<tr>
<td>Get_Date()</td>
</tr>
</tbody>
</table>

Figure 8. Dates Class

c. Time Class

Description: The Time Class contains a type Tim (Hours, Minutes, Seconds).

The Class includes Functions and Procedures to Get a Time from the computer system or GPS, as well as, return all the Time values and set all the Time values.
d. Latitude/Longitude Class

Description: The Lat_Long Class contains a type Latitude (Degrees, Minutes, Seconds, Sign) and type Longitude (Degrees, Minutes, Seconds, Sign). The Class includes functions to convert from Nautical Miles to Kilometers, Yards to Kilometers, Yards to Nautical Miles and the converse functions as well. Also included in this class are two unique procedures. The first, Calculates a Latitude and Longitude given a bearing and range from a known Lat/Long Position. This Procedure also returns the back bearing based on trigometric formulas that take into account the curvature of the earth. The other Procedure takes two known Lat/Long Positions and then calculates the distance between them and the forward and back bearings. The Class also contains the basic set and get procedures/ functions for each Latitude and Longitude type.
Figure 10. Lat_Long Class

e. Hit Class

Description: The Hit class contains all of the data necessary when a hit is taken (a bearing and range to a contact from a radar scope). A hit type includes Bearing, Range, Latitude, Longitude, Date, Time, Ownship Course and Speed, Target Course and Speed, and
Target Angle. Hit Class contains all the procedures necessary to set and get data from each element of the Hit type.

**Hit Class**

- **Hit**: Record
  - **Bear**: Degree = 0.0
  - **Rng**: Real = 0.0
  - **Lat**: Latitude
  - **Lon**: Longitude
  - **Dat**: Date = Get_Date_From_System
  - **Time**: Tim = Get_Time_From_System
  - **Target_Cse**: Degree = 0.0
  - **OwnShip_Cse**: Degree = 0.0
  - **Target_Speed**: Speed = 0.0
  - **OwnShip_Speed**: Speed = 0.0

- **Set_Hit_Time()**
- **Set_Hit_Hour()**
- **Set_Hit_Min()**
- **Set_Hit_Sec()**
- **Get_Hit_Time()**
- **Get_Hit_Hour()**
- **Get_Hit_Min()**
- **Get_Hit_Sec()**
- **Set_Hit_Date()**
- **Set_Hit_Day()**
- **Set_Hit_Month()**
- **Set_Hit_Year()**
- **Get_Hit_Date()**
- **Get_Hit_Day()**
- **Get_Hit_Month()**
- **Get_Hit_Year()**
- **Set_Hit_Lat()**
- **Get_Hit_Lat()**
- **Set_Hit_Lon()**
- **Get_Hit_Lon()**
- **Set_Hit_Bearing()**
- **Get_Hit_Bearing()**
- **Set_Hit_Range()**
- **Get_Hit_Range()**
- **Set_OwnShip_Cse()**
- **Get_OwnShip_Cse()**
- **Set_Target_Cse()**
- **Get_Target_Cse()**
- **Set_OwnShip_Speed()**
- **Get_OwnShip_Speed()**
- **Set_Target_Speed()**
- **Get_Target_Speed()**
- **Calc_Hit_Lat_Long()**

---

**f. Track Class**

**Description**: The Track Class contains all of the data necessary to track a contact. The track type includes Track Number, Track Ident, Track Course and Speed, CPA Bearing, CPA Range, CPA Time, a Hit Count and a List of Hits. Track includes
functionality to set and get all of the elements of hit type. Track also maintains an array of 10,000 elements that are of type track. This is the memory allocation for all of the tracks prior to being saved onto a harddisk or storage device.

**Track Class**

- track : Record
- Point : Array of Hit
- Track_Number : Integer 1100..1100
- Track_ID : string
- Track_Cse : Degree
- Track_Spd : Speed
- CPA_Bearing : Degree
- CPA_Range : Real
- CPA_Time : Tim
- Hit_List : Point
- Hit_Count : Integer

**Figure 12. Track Class**

**g. OwnShip Class**

*Description:* OwnShip Class manages and maintains all information concerning OwnShip. This includes Number, Identification, Course and Speed, Latitude, and Longitude. The OwnShip Class contains all of the required Get and Set functions and procedures necessary to modify and retrieve type data. OwnShip Class also retrieves Latitude and Longitude Information from either manual keyboard entry or from a GPS port (ex. COM1). This is based upon the user-selected mode.
h. GPS Class

Description: This Class manages the interface between the GPS System and the Computer system. This Class retrieves GPS information from the designated port (ex. COM1) then parses and stores this information into usable formats, i.e. Latitude, Longitude, Date, Time, etc.

i. Speed Class

Description: Defines subtype Speed that is a Real type with digits 1 range 0.0 to 130.0 This maximum values was chosen based on the simple fact that our design of this system is intended to track surface vessels, not aircraft, etc.
j. **Realnum Class**

*Description:* Defines subtype Real that is digits 12. This class instantiates Generic_Elementary_Functions (Real) allowing the Real Type to use all of the Generic Elementary Functions.

![Realnum Class Diagram](image)

Figure 15. Realnum Class

k. **Degree Class**

*Description:* Defines subtype Degree that is a Real type with digits 1 range 0.0 to 359.9

![Degree Class Diagram](image)

Figure 16. Degree Class

l. **Radar Class**

*Description:* This is a future Class that will be designed to handle Radar input of track information and manage the interface between the two systems.

m. **Network Class**

*Description:* Manages all of the network traffic and sequence of events required to pass data over a small network.
n. CPA Class

Description: This Class handles all Closest Point of Approach (CPA) calculations and then plots those DRM, SRM, and Associated Speed Triangle and CPA information to the Moboard, DRT, or specified view.

<table>
<thead>
<tr>
<th>CPA Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed_Scale :CONSTNAT Realnum := 5.0</td>
</tr>
<tr>
<td>Number_O_Circles : CONSTANT Integer := 11;</td>
</tr>
<tr>
<td>One_Degree : Realnum.Real</td>
</tr>
<tr>
<td>CPA_Drawing_Area : Sketchpad.Gtk_Sketchpad</td>
</tr>
<tr>
<td>X_Origin</td>
</tr>
<tr>
<td>Y_Origin</td>
</tr>
<tr>
<td>CPA_Alert_Distance : Realnum</td>
</tr>
<tr>
<td>Fair Xpos and Fair Ypos</td>
</tr>
<tr>
<td>Fair First Bearing, Fair Second Bearing : Utilities.My_Degree</td>
</tr>
<tr>
<td>Calculate_SRM()</td>
</tr>
<tr>
<td>Finish_Speed_Triangle()</td>
</tr>
</tbody>
</table>

Figure 17. CPA Class

o. Moboard Class

Description: The Moboard Class displays OwnShip and Track Class Data in a relative 0 to 360 degree coordinate system scale based upon the speeds of the vectors.
Figure 18. Moboard Class

p. **MainScreen-Pkg Class**

Description: This Class handles all the Packages for the Mainscreen.
Figure 19. MainScreen-Pkg Class

q. **MainScreen-Pkg-Callbacks Class**

Description: This Class handles all the Callbacks for the Main Screen Packages.
r. Sketchpad Class

Description: This class is similar to the name, it’s a Sketch Pad that allows the programmer to draw in the background and then load the information to the front of the view.
s. **Utilities Class**

Description: This Class is designed to encompass all generic functions and procedures, such as converting real and integer numbers to strings, as well as any generic type definitions that multiple classes require to use as part of the program.

![Utilities Class](image)

Figure 22. Utilities Class

1. **File IO Class**

Description: This Class is designed to handle all data recovery/restoration of track, hit, and ownership information. The database maintains the last known state in three (3) separate files. Should the system shutdown or crash, the program will automatically load the last stored information from the Latest files kept under File_IO Class.
v. Historical IO Class

Description: This Class is designed to save all data for every change or update made to ownership, track, or hit information. The database is a sequential database that has a sentinel designating the type of data to be stored. This will allow for future expansion for data re-play, reconstruction of events, and/or data playback for mishaps, exercise evaluations, etc.
w. **Wind Class**

Description: This Class is designed to calculate True wind and Desired wind and then plot the result onto the Moboard, DRT, or specified view that the user desires.

![Wind Class Diagram](image)

Figure 25. Wind Class

3. **Deployment Diagram**

![Deployment Diagram](image)

Figure 26. Deployment Diagram
4. **GtkAda and GNAT**

There are several reasons we chose to use GtkAda and GNAT compilers. First off, as students with no funding, both of these compilers are free, a very attractive quality. Secondly, part of our research was to design a computer program that was hardware and software (Operating System) independent, thus portable. GtkAda is a high level portable graphical toolkit based on the gtk+ toolkit and one of the official GNU toolkits. Additionally, GtkAda uses Ada95 features and supports Object Orientation. Another attractive feature of GtkAda is that it supports OpenGL.

![Figure 27. GtkAda layered structure](image)

There is no guarantee that the DOD will be using Windows NT or Linux, or any other operating system 5 years from now. Our program is designed to be and is hardware and software independent. This is an attractive feature that gives the DOD flexibility in operating system procurement, as well as, makes our code more maintainable and robust. Our code can be run on all of the following platforms:

- Linux/x86
- Linux/sparc
- Linux/ppc
- Solaris/sparc
- Solaris/x86
- Dec Unix
- SGI IRIX 6.5
- HP/UX
- NT 4.0
- Windows 2000
5. GNU Visual Debugger (GVD)

GVD is a graphical front-end for the text based GDB debugger that is provided with the GNAT package. The GNU Visual Debugger enables you to see what is going on inside the program while it is executing, or what the program is doing when it crashes and where it crashes. During the preliminary stages of our research, GVD was not available for use. This required us to do Ada.Text_IO style debugging. If there were runtime errors, it was a requirement to insert dos based Print Lines, or Ada.Text_IO.Put_Line statements into the code in order to track down the procedure or function that the program was halting/crashing in. This was a very time consuming and tedious method of debugging. With the release of GVD, debugging became a much easier task. The debugger itself is very user friendly, it’s style is consistent with with any standard debugger that an accomplished programmer would be familiar with. GVD features of note include Open Core Dump, Edit Source, Attach to process, Detach process, Show call stack, show local variables, show arguments, show registers, examine memory, and simple task analysis.

GVD is a very simple debugger and does not take a lot of time to become familiar enough with to use. GVD has many advanced features like the ability to list the threads an executable is currently running by internal identifier, name, and status. Then the user can click on the thread and change the context variables, call stack, or source file.

The Call Stack Window gives you a list of frames that correspond to the current stack of execution for the current thread or task. By simple mouse click you can choose what information you want to display in the call stack window. By default the subprogram and parameters are displayed. The options to display frame number, program counter, and file location are available.

Overall, GVD was a very robust and usable tool. Attractive to students because of the wonderful price tag of being free, the quality of the product is on a much higher scale then the typical Freeware that floats around on the Internet. Our experience was
very positive. The only negative comment is that when using GtkAda graphics, it is sometimes difficult to step through line by line while within the graphics portions of the code and it often became necessary to run to specific points in the code, and then do step-by-step debugging. We would recommend this Visual Debugger front-end to anyone doing large or small Ada programs using the GNAT compiler.

Figure 28. Screenshot of GVD Version 1.2.1
IV. CONCLUSION

A. SUMMARY

The project began with a very simple idea. Take a proven method, known and understood by all Surface Warfare Officers and make the means by which the process is executed better. The Moboard is based on technology of the 1940s, we know the process works, and it has been time-tested over generations of sailors. The method of using pencil and paper is littered with points where even the most capable naval officer can make a mistake that can result in less than accurate answers. The paper-base process takes time to complete, and when given more then one contact to track, the problem becomes increasingly harder, especially when dealing with different range scales on a single Moboard.

What our program has done is change the medium by which the traditional Moboard is calculated. There is no doubt, that given a head-to-head match-up our program as accurate, if not more accurate then the paper version, and much faster. In our current version there is limited Artificial Intelligence that alerts the user when a contact will be coming closer then the Captains set standing order distance, or when a track is Constant Bearing Decreasing Range (CBDR). The project has the potential to grow into a much more intelligent program and there is a host of future work to be studied and worked on.

B. RECOMMENDATIONS FOR FUTURE WORK

Future work will include a set of additional views (an extensibility of our design). There are extensive areas of follow-on work. For this program to truly meet the needs of tomorrow’s navy, continued and future research is required. We expect the project will evolve to incorporate (but not limited to) the following technologies:
1. **GPS Integration**

Our current implementation is in the development stages of receiving GPS via a computer COM port. These receivers are inexpensive and offer a high degree of accuracy. One of the tenants of our software design is to maintain platform portability, by remaining hardware and operating system independent. To date, we have been unable to design the system to be completely operating system independent due to the differences between how Windows NT based systems and Linux based systems handle COM ports. Future work will include building drivers and program integration that supports USB interfaces and designing for portability. Additionally, work on integrating shipboard based GPS systems is an additional avenue that could also be pursued in future research. Additionally, the integration of Wireless GPS connectivity in a shipboard environment (where a hard mounted GPS receiver broadcasts to a mobile platform and continuously updates a ship’s position to the program) would make the system entirely mobile within the confines of the ship.

2. **Radar Data Integration**

When you discuss radar integration with any system, it is important to design your system to be able to integrate with any current or future radar. Our modular design will enable us to integrate easily and quickly with new and existing radar systems. There are two thought processes when we discuss radar integration. One system would maintain the “man in the loop” principle, and the other is a more automated system, similar to the SPY-1 radar system.

Considering radar integration with a man in the loop, the vision is a radar repeater similar to the SPA-25 where the actual “blip” or radar return is displayed on the scope. Our Moboard program would be the overlay on the repeater scope. When a new track is seen on the scope, the JOOD/OOD or whoever is designated as the operator takes a plastic pen device and clicks on the screen “New Contact” and then takes the pen and touches the screen where the radar video is. A hit symbol and all the info associated with that point is automatically displayed and it automatically calculates associated information. Then to track the contact, the operator simply selects the track they are interested in, and then marks a new position of the radar video. This automatically gives
the operator warnings and alerts for CBDR, vessels of high interest, rules of the road, and any vessel projected to be within the Captains standing distance.

The second avenue that radar integration could follow is a more automated system where the system detects, based on some level of sensitivity, a new contact, and automatically plots and can calculates all associated information. This could also have automatically generated alerts and warnings. This could be an option on the program, either in manual mode or auto mode. The auto mode would still have the ability to add hits manually. The potential for extensive follow work exists in the area of Radar integration.

3. Touch screen displays

In a similar fashion, part of our usability studies resulted with feedback from Surface Warfare Officers who felt that the system and the idea were a great start, but to make the system even better we could replace the mouse or touch pad with a touch-screen. This would make the system more usable and easier to operate in “at-sea” conditions.

We envision a future system being completely touch-screen based. In this system the operator or user would not have to operate a mouse or touch pad. There is considerable work in applying touch-screen technology. The current system is modular enough that reprogramming would mesh well with the integration of touch-screens. The future work would include integrating touch-screens, developing the callbacks and new code to handle the different events from a touch-screen. Additionally, linking this work and technology with the integration of the radar video. The ultimate vision is a flat-screen display above the Commanding Officers chair on the bridge where the CO can instantly look up and get critical safety information related to the maneuvering of his/her ship. If the CO desires to get additional information or tunnel deeper into the problem, then he/she can simply pull the information with a touch of the screen.

4. Wireless LAN connectivity

Another area of work is Wireless LAN connectivity. Specifically we are talking about the connectivity of wireless data transfer internal to the ship. The work would be related to the Wireless portion of the LAN. That is, ensure that the LAN, baring hardware
failures, from a software viewpoint is error free and maintains continuous connectivity. For, example, if a Commanding Officer has a portable device, and the OOD sends a signal to notify the Captain that there is a contact of interest or contact report for him, the system must have a level of reliability that ensures that regardless of where the CO is onboard the ship, he will be guaranteed to receive those messages with some level of confidence. Additionally, there is work in design, coding, and implementation of the Ada modules that would handle the Wireless Network traffic and functionality.

5. Voice recognition technology

Extensive work remains in the area of Voice recognition technology. If in the future, a voice recognition system is implemented where the Conning Officer can wear a headset and give commands like “Right Standard Rudder” and the steering system simply responds, then this type of system must have 100 percent reliability and be zero error prone.

On a less critical level, the Automatic Deck Log could utilize voice recognition technology. In less scaled version of this design, the OOD or Conning officer would give the commands and voice recognition technology would automatically enter speed and course changes into the Automatic Deck Log as well as the Digital Moboard Computer. The voice commands would not be connected to the steering system directly, a Helmsman would still be “in the loop” per say, and respond to the commands and execute the order.

6. Mobile headset/communications

This area of research is closely related to the voice recognition technology, but focuses more on the area of headset usability, environment analysis of headset technology, and the suitability of those products in the shipboard environment. Additionally, research in the area of wireless headset communications, the reliability of those communications, maximum noise and distortion levels allowed while still maintaining the clarity and reliability required for continuous uninterrupted communications is required. Extensive work must be done testing this type of system to validate and verify its reliability. One of the most important aspects of maneuvering the ship is safety and reliability of any system. Before implementing this type of system, the
system must be thoroughly tested to demonstrate adequate data integrity and reliability to allow safety critical information to be transmitted via wireless communications.

7. **Automated Deck Log**

This research is a direct follow-on from our Digital Moboard program. The Deck Log should be written in Ada using the GNAT compiler and the GtkAda toolkit so that our original design of hardware and operating system portability, maintainability, extensibility are maintained with the additional functionality. The Automated Deck Log could be integrated with our Digital Moboard program in the following aspects. The Automated Deck Log would be a separated entity, a separate program that would maintain a log of events. The information could be entered via keyboard input, voice data entry, or stylis touch-screen type technology. The means by which the data or log entries are made is part of the follow-on research, but the design of the program should be modular enough that it does not matter how the data gets there, a standard interface and common protocol should be used. When either a speed or course is made in the Digital Moboard, a message would be sent to tell the Automatic Deck Log to make an entry, for example, “Course 183 Speed 13 knots.” Customized entries could be made in the log. During watch turnover the oncoming and offgoing OODs could digitally sign with the stylis on the screen. At the end of the day, the deck logs would be backed up for permanent archives, and both the paper and digital version would be signed by the Commanding Officer.

8. **Palm Pilot/CE devices providing information on demand**

Imagine that you’re the OOD underway; you have a contact that you have done a moboard on, and you need to call the Captain. The Captain is down in Engineering doing a Walkthrough, you have called, done an announcement on the 1MC and still have not heard from the Captain. What do you do? The vision we forsee is a portable device that the Captain can carry on his belt with a graphical display that would display a mirror image of what the OOD sees on the bridge. When a contact of interest requires the Captain’s attention, his/her portable device vibrates and/or beeps until he reviews the contact of interest and presses his approval of the OODs recommendation or sends the OOD a message saying, “Do this instead.” Now that the CO has an instantaneous
graphical picture of the situation on the bridge; he/she can call the OOD and discuss the situation further if he/she feels necessary, or make his/her way to the bridge.

Research should include a comprehensive study on types of portable devices, from both a usability point of view to a functionality point of view. Additionally, research should be linked with the Wireless connectivity of those devices on a shipboard environment.

9. **Integrated multiple views (FalconView, Heads-up displays, etc.)**

Research in the area of developing and/or integrating multiple views of the track information that we have in our system could include: Integrating FalconView style programs. These programs will use the Latitude and Longitude information that Digital Moboard contains and display the track hits within FalconView so that the user can see the geographical location on a NIMA chart of both your own ship as well as any contacts that you are tracking via radar. Additional views or uses remain open to research; such studies include usability work on the best way to present data to the ship driver. Additional research can be done in the area of Heads-up display screens, 3-D Graphical representations, Holographic data representations, etc.

10. **Ada enabled Applets for browsers exchange**

Future work in this area would include enabling Ada Applets for Web based applications. This would include, the ability to pull information up from specific platforms to a web browser as well as tunnel down into the data that specified platforms have. This would allow the Battle Group Commander to view the “picture” that each ship has, or consolidate the data from each platform and display it in a web-based format.

11. **Artificial Intelligent Maneuvering Modules**

Our program contains the functionality to alert the OOD and CO when a contact will be within a set distance based upon the Commanding Officers Standing Orders. Future work includes more expert systems that generate signals for the OOD and/or CO that the designated contact will be within minimum distances set combined with maneuvering recommendations. The program should include functionality that gives the OOD recommendations based upon a standared Rules of the Road library. This library will alert the OOD/CO of important Rules of the Road and give recommended course and
speed changes based upon those rules and the Captains Standing Orders. The default being, maneuver the ship to maintain minimum set Closets Point of Approach (CPA). The computer program will also alert the OOD/CO if a course and/or speed change will affect the CPA of other contacts adversely or affect the safe navigation of the ship. These enhancements will be a guide/aid to the OOD/CO helping ensure safe navigation at sea thereby preventing collisions resultant to human-error, fatigue, or oversight.
APPENDIX A. DIGITAL MOBOARD CODE

This appendix contains the object-oriented code of the Digital Moboard program. Included in this appendix are all of the .ads files used in our Object-orientated design.

The current version of the program is divided into twenty-two classes. Enclosed are the .ads files and the Main program start point Navigator.adb

A-1 DATES.ADS

--**********************************************************************
--| FILE: dates.ads
--| AUTHOR: Joey L. Frantzen, Naval Post Graduate School
--| LAST MODIFIED: 11 September 2001
--| OPERATING ENVIRONMENT: Windows 2000(Designed to be O/S Independent)
--| COMPILER: GNAT 3.13p
--| DESCRIPTION: This class is an upgrade taken from Ada 95 Problem Solving and
--| Specification for package to represent dates, Orig Author Michael B. Feldman
--| The George Washing University wrote Get_Date_From_System(everything else is
--| original work.
--| INPUTS: Ada.Calendar
--| OUTPUTS: Outputs based on what Functions are requested and run.
--| Process: None to note
--| Assumptions: N/A
--| Warnings: None

39
WITH Ada.Calendar;

PACKAGE dates IS

TYPE Date_Type IS PRIVATE;

SUBTYPE Month_Number IS Ada.Calendar.Month_Number;
SUBTYPE Year_Number IS Ada.Calendar.Year_Number;
SUBTYPE Day_Number IS Ada.Calendar.Day_Number;

Date_Error : EXCEPTION;

--**********************************************************************

FUNCTION Get_Date_String_From_Number(M : IN Integer) RETURN String;

--**********************************************************************
--| Function: Get_Date_From_System

--| Input: Year, Month, Day

--| Output: Returns a Date_Type; analogous to Ada.Calendar.Clock

--| Warnings: Date_Error if the year, month, day triple do not

--| form a valid date (Feb 30th, for example)

--| Analogous to Ada.Calendar.Time_Of

FUNCTION Get_Date_From_System RETURN Date_Type;

--| Procedure: Set_Date

--| Input: Year_Number, Month_Number, Day_Number, Date_Type

--| Output: Date_Type

--| Description: Sets Year, Month, and Day into Date_Type

--| Warnings: None

PROCEDURE Set_Date( D : IN Day_Number; M : IN Month_Number;

                     Y : IN Year_Number;

                     Dat: IN OUT Date_Type);

--| Procedure: Set_Month

--| Input: Month_Number, Date_Type

--| Output: Date_Type
--| Description: Sets Year into Date_Type

--| Warnings: None

--**********************************************************************

PROCEDURE Set_Month(M : IN Month_Number; Dat : IN OUT Date_Type);

--**********************************************************************

--| Procedure: Set_Day

--| Input:  Day_Number, Date_Type

--| Output: Date_Type

--| Description: Sets Day into Date_Type

--| Warnings: None

--**********************************************************************

PROCEDURE Set_Day(D : IN Day_Number; Dat : IN OUT Date_Type);

--**********************************************************************

--| Procedure: Set_Year

--| Input:  Year_Number, Date_Type

--| Output: Date_Type

--| Description: Sets Year into Date_Type

--| Warnings: None

--**********************************************************************

PROCEDURE Set_Year(Y : IN Year_Number; Dat : IN OUT Date_Type);
--| Function: Get_Date
--| Input: Date_Type
--| Output: Date_Type
--| Description: Returns a Date_Type value
--| Warnings: None

FUNCTION Get_Date(D : Date_Type) RETURN Date_Type;

--| Function: Get_Month
--| Input: Date_Type
--| Output: Month_Number
--| Description: Returns a integer value
--| Warnings: None

FUNCTION Get_Month(D : Date_Type) RETURN integer;

--| Function: Get_Day
--| Input: Date_Type
--| Output: Dat_Number
--| Description: Returns a integer value
--| Warnings: None

FUNCTION Get_Day(D : Date_Type) RETURN integer;
FUNCTION Get_Day(D : Date_Type) RETURN integer;

--******************************************************************************

--| Function: Get_Year
--| Input:  Date_Type
--| Output: Year_Number
--| Description: Returns a integer value
--| Warnings: None
--******************************************************************************

FUNCTION Get_Year(D : Date_Type) RETURN integer;

PRIVATE

TYPE Date_Type IS RECORD
    Month : Month_Number := Ada.Calendar.Month(Ada.Calendar.Clock);
    Day : Day_Number := Ada.Calendar.Day(Ada.Calendar.Clock);
    Year : Year_Number := Ada.Calendar.Year(Ada.Calendar.Clock);
END RECORD;

END dates;
PACKAGE times IS

TYPE Time_Type IS PRIVATE;
--| Function: Get_Time_Of_Day()

--| Input: None

--| Output: Current Time from Ada.Calendar.Clock in a Time_Type

--| Process: Finds today's Time and returns it as a record of type Time

--| Today's Time is gotten from PACKAGE Ada.Calendar

FUNCTION Get_Time_Of_Day RETURN Time_Type;

--| Procedure: Set_Time

--| Input: Hour, Minutes, Seconds, Time_Type

--| Output: Time_Type

--| Description: Sets Hour, Minutes, and Seconds into Time_Type

--| Warnings: None

PROCEDURE Set_Time(Hr : IN Integer; Min : IN Integer; Sec : IN Integer;
T: IN OUT Time_Type);

--| Procedure: Set_Hours

--| Input: Hour, Time_Type

--| Output: Time_Type

--| Description: Sets Hours into Time_Type

--| Warnings: None
PROCEDURE Set_Hours(Hr : IN Integer; T : IN OUT Time_Type);

PROCEDURE Set_Mins(Min : IN Integer; T : IN OUT Time_Type);

PROCEDURE Set_Secs(Sec : IN Integer; T : IN OUT Time_Type);

FUNCTION Get_Time(T : Time_Type);
FUNCTION Get_Time(T : Time_Type) RETURN Time_Type;

FUNCTION Get_Hours(T: Time_Type) RETURN Integer;

FUNCTION Get_Mins(T : Time_Type) RETURN Integer;
FUNCTION Get_Secs(T : Time_Type) RETURN Integer;

PRIVATE

TYPE Time_Type IS RECORD
  Hours   : Integer RANGE 0..24 := 0;
  Mins    : Integer RANGE 0..59 := 0;
  Secs    : Integer RANGE 0..59 := 0;
END RECORD;

END times;

A-3 HIT.ADS

--**********************************************************************
--| Function: Get_Secs
--| Input:  Time_Type
--| Output: Integer(Seconds)
--| Description: Returns an Integer from Time_Type(Seconds)
--| Warnings: None
--**********************************************************************

FUNCTION Get_Secs(T : Time_Type) RETURN Integer;

PRIVATE

TYPE Time_Type IS RECORD
  Hours   : Integer RANGE 0..24 := 0;
  Mins    : Integer RANGE 0..59 := 0;
  Secs    : Integer RANGE 0..59 := 0;
END RECORD;

END times;

--**********************************************************************
--| FILE: Hit.ads
WITH lat_long;
WITH dates;
WITH times;
WITH Realnum;
WITH degrees;
WITH speeds;
WITH ownship;

PACKAGE hit IS
TYPE Hits IS PRIVATE;

--TIME PROCEDURES and FUNCTIONS *******************************
--**********************************************************************
--| Procedure: Set_Hit_Time
--| Input:  Hour, Minute, Seconds(all type Integer), Hit_Type
--| Output: Hit_Type
--| Description: Saves Hour, Minute and Seconds into Hit_Type.T_ime
--**********************************************************************
PROCEDURE Set_Hit_Time(Hr : IN Integer; Min : IN Integer; Sec : IN Integer;
                        H : IN OUT Hits);

--**********************************************************************
--| Procedure: Set_Hit_Time
--| Input:  Time_Type, Hit_Type
--| Output: Hit_Type
--| Description: Saves Time_Type into Hit_Type.T_ime
--**********************************************************************
PROCEDURE Set_Hit_Time(T_ime : IN times.Time_Type; H : IN OUT Hits);

--**********************************************************************
--| Procedure: Set_Hit_Hours

51
PROCEDURE Set_Hit_Hours(Hr : IN Integer; H : IN OUT Hits);

PROCEDURE Set_Hit_Mins(Min : IN Integer; H : IN OUT Hits);

PROCEDURE Set_Hit_Secs(Sec : IN Integer; H : IN OUT Hits);

--| Function: Get_Hit_Time
--| Input: Hit_Type
--| Output: Time_Type
--| Description: Returns Hit_Type.T ime Value of inputed Hit_Type

FUNCTION Get_Hit_Time(H : Hits) RETURN times.Time_Type;

--******************************************************************************
--| Function: Get_Hit_Hours
--| Input: Hit_Type
--| Output: Integer
--| Description: Returns Integer Value(Hours) of inputed Hit_Type.T ime

FUNCTION Get_Hit_Hours(H: Hits) RETURN Integer;

--******************************************************************************
--| Function: Get_Hit_Mins
--| Input: Hit_Type
--| Output: Integer
--| Description: Returns Integer Value(Minutes) of inputed Hit_Type.T ime

FUNCTION Get_Hit_Mins(H : Hits) RETURN Integer;

--******************************************************************************
--| Function: Get_Hit_Secs

FUNCTION Get_Hit_Secs
-- Input: Hit_Type
-- Output: Integer
-- Description: Returns Integer Value(Seconds) of inputed Hit_Type.T ime

FUNCTION Get_Hit_Secs(H : Hits) RETURN Integer;

--DATE PROCEDURE and FUNCTIONS

-- Procedure: Set_Hit_Date
-- Input:  Day_Number, Month_Number, Year_Number, Hit_Type
-- Output: Hit_Type
-- Description: Saves Day, Month and Year into Hit_Type.Dat

PROCEDURE Set_Hit_Date(D : IN dates.Day_Number; M : IN dates.Month_Number;
                        Y : IN dates.Year_Number; H: IN OUT Hits);

-- Procedure: Set_Hit_Date
-- Input: Date_Type, Hit_Type
-- Output: Hit_Type
-- Description: Saves Date_Type into Hit_Type.Dat
PROCEDURE Set_Hit_Date(D : IN dates.Date_Type; H: IN OUT Hits);

PROCEDURE Set_Hit_Month(M : IN dates.Month_Number; H : IN OUT Hits);

PROCEDURE Set_Hit_Day(D : IN dates.Day_Number; H : IN OUT Hits);

PROCEDURE Set_Hit_Year(D : IN dates.Year_Number; H : IN OUT Hits);

-- Procedure: Set_Hit_Month
-- Input: Month_Number, Hit_Type
-- Output: Hit_Type
-- Description: Saves Month into Hit_Type.Dat

PROCEDURE Set_Hit_Day(D : IN dates.Day_Number; H : IN OUT Hits);

PROCEDURE Set_Hit_Year(D : IN dates.Year_Number; H : IN OUT Hits);

-- Procedure: Set_Hit_Year
-- Input: Year_Number, Hit_Type
-- Output: Hit_Type
-- Description: Saves Year into Hit_Type.Dat
PROCEDURE Set_Hit_Year(Y : IN dates.Year_Number; H : IN OUT Hits);

FUNCTION Get_Hit_Date(H : Hits) RETURN dates.Date_Type;

FUNCTION Get_Hit_Month(H : Hits) RETURN dates.Month_Number;

FUNCTION Get_Hit_Day(H : Hits) RETURN dates.Day_Number;
FUNCTION Get_Hit_Day(H : Hits) RETURN dates.Day_Number;

FUNCTION Get_Hit_Year(H : Hits) RETURN dates.Year_Number;

PROCEDURE Set_Hit_Latitude(D : IN Integer; M : IN Integer; S : IN Integer; Sign : IN Character; H: IN OUT Hits);

PROCEDURE Set_Hit_Latitude(D : IN Integer; M : IN Integer; S : IN Integer; Sign : IN Character; H: IN OUT Hits);
PROCEDURE Set_Hit_Latitude(Lat : IN lat_long.Latitude; H: IN OUT Hits);

PROCEDURE Set_Hit_Lat_Deg(D : IN Integer; H : IN OUT Hits);

PROCEDURE Set_Hit_Lat_Min(M : IN Integer; H : IN OUT Hits);

PROCEDURE Set_Hit_Lat_Sec
PROCEDURE Set_Hit_Lat_Sec(S : IN Integer; H : IN OUT Hits);

PROCEDURE Set_Hit_Lat_Sign(Sign : IN Character; H : IN OUT Hits);

FUNCTION Get_Hit_Latitude(H : Hits) RETURN lat_long.Latitude;

FUNCTION Get_Hit_Lat_Deg
--| Input:  Hit_Type
--| Output: Latitude Degree(Integer)
--| Description: Returns Latitude Degree Value of inputed Hit_Type.Lat

FUNCTION Get_Hit_Lat_Deg(H : Hits) RETURN Integer;

--| Function: Get_Hit_Lat_Min
--| Input:  Hit_Type
--| Output: Latitude Minute(Integer)
--| Description: Returns Latitude Minute Value of inputed Hit_Type.Lat

FUNCTION Get_Hit_Lat_Min(H : Hits) RETURN Integer;

--| Function: Get_Hit_Lat_Sec
--| Input:  Hit_Type
--| Output: Latitude Seconds(Integer)
--| Description: Returns Latitude Seconds Value of inputed Hit_Type.Lat

FUNCTION Get_Hit_Lat_Sec(H : Hits) RETURN Integer;

--| Function: Get_Hit_Lat_Sign
FUNCTION Get_Hit_Lat_Sign(H : Hits) RETURN Character;

PROCEDURE Set_Hit_Longitude(D : IN Integer; M : IN Integer; S : IN Integer; Sign : IN Character; H: IN OUT Hits);

PROCEDURE Set_Hit_Longitude(D : IN Integer; M : IN Integer; S : IN Integer; Sign : IN Character; H: IN OUT Hits);

--| Procedure: Set_Hit_Longitude
--| Input: Longitude Type, Hit_Type
--| Output: Hit_Type
--| Description: Saves Longitude Type into Hit_Type.Lat
PROCEDURE Set_Hit_Longitude(Lon : IN lat_long.Longitude;
                            H: IN OUT Hits);

PROCEDURE Set_Hit_Long_Deg(D : IN Integer; H : IN OUT Hits);

PROCEDURE Set_Hit_Long_Min(M : IN Integer; H : IN OUT Hits);

PROCEDURE Set_Hit_Long_Sec(S : IN Integer; H : IN OUT Hits);
PROCEDURE Set_Hit_Long_Sec(S : IN Integer; H : IN OUT Hits);

PROCEDURE Set_Hit_Long_Sign(Sign : IN Character; H : IN OUT Hits);

FUNCTION Get_Hit_Longitude(H : Hits) RETURN lat_long.Latitude;

FUNCTION Get_Hit_Long_Deg(H : Hits) RETURN lat_long.Longitude;
|-- Description: Returns Longitude Degree Value of inputed Hit_Type.Lon

FUNCTION Get_Hit_Long_Deg(H : Hits) RETURN Integer;

|-- Function: Get_Hit_Long_Min

|-- Input: Hit_Type

|-- Output: Longitude Minute(Integer)

|-- Description: Returns Longitude Minute Value of inputed Hit_Type.Lon

FUNCTION Get_Hit_Long_Min(H : Hits) RETURN Integer;

|-- Function: Get_Hit_Long_Sec

|-- Input: Hit_Type

|-- Output: Longitude Seconds(Integer)

|-- Description: Returns Longitude Seconds Value of inputed Hit_Type.Lon

FUNCTION Get_Hit_Long_Sec(H : Hits) RETURN Integer;

|-- Function: Get_Hit_Long_Sign

|-- Input: Hit_Type

|-- Output: Longitude Sign(Character)
--| Description: Returns Longitude Hemisphere Sign Value of inputed Hit_Type.Lon

FUNCTION Get_Hit_Long_Sign(H : Hits) RETURN Character;

--SET and GET BEARING PROCEDURE and FUNCTIONS***************

--| Procedure: Set_Hit_Bearing
--| Input: Bearing, Hit_Type
--| Output: Hit_Type
--| Description: Sets Hit Bearing in Hit_Type to Inputed Value

PROCEDURE Set_Hit_Bearing(Bear : IN degrees.Degree; H : IN OUT Hits);

--| Function: Get_Hit_Bearing
--| Input: Hit_Type
--| Output: Bearing (Realnum.Real)
--| Description: Returns Bearing Degree Value of inputed Hit_Type

FUNCTION Get_Hit_Bearing(H : Hits) RETURN Realnum.Real;

--SET and GET RANGE PROCEDURE and FUNCTIONS***************
PROCEDURE Set_Hit_Range(Rng : IN Realnum.Real; H : IN OUT Hits);

FUNCTION Get_Hit_Range(H : Hits) RETURN Realnum.Real;

PROCEDURE Set_Hit_OwnShip_Cse(Course, Hit_Type : IN);
PROCEDURE Set_Hit_OwnShip_Cse(Course : in degrees.Degree; H : IN OUT Hits);

FUNCTION Get_Hit_OwnShip_Cse(H : Hits) RETURN Realnum.Real;

PROCEDURE Set_Hit_Target_Cse(Course : in degrees.Degree; H : IN OUT Hits);

FUNCTION Get_Hit_Target_Cse(H : Hits) RETURN Realnum.Real;

PROCEDURE Set_Hit_Target_Cse(Course : in degrees.Degree; H : IN OUT Hits);
-- Output: Target Course(Realnum.Real)
-- Description: Returns Target Course Value of inputed Hit_Type

FUNCTION Get_Hit_Target_Cse(H : Hits) RETURN Realnum.Real;

--SET and GET OWNSHIP_SPEED PROCEDURE and FUNCTIONS**************

-- Procedure: Set_Hit_OwnShip_Speed
-- Input: Speed, Hit_Type
-- Output: Hit_Type
-- Description: Sets OwnShip Speed into Hit_Type to Inputed Value

PROCEDURE Set_Hit_OwnShip_Speed(Spd : IN speeds.Speed; H : IN OUT Hits);

-- Function: Get_Hit_OwnShip_Speed
-- Input: Hit_Type
-- Output: OwnShip Speed(Realnum.Real)
-- Description: Returns OwnShip Speed Value of inputed Hit_Type

FUNCTION Get_Hit_OwnShip_Speed(H : Hits) RETURN Realnum.Real;

--SET and GET HIT COUNTER PROCEDURE and FUNCTIONS**************

68
--**********************************************************************
--| Procedure: Set_Hit_Counter
--| Input: Counter, Hit_Type
--| Output: Hit_Type
--| Description: Sets Counter into Hit_Type to Inputed Value
--**********************************************************************
PROCEDURE Set_Hit_Counter(Counter : IN integer; H : IN OUT Hits);

--**********************************************************************
--| Function: Get_Hit_Counter
--| Input: Hit_Type
--| Output: Hit Counter(Integer)
--| Description: Returns Hit Counter Value of inputed Hit_Type
--**********************************************************************
FUNCTION Get_Hit_Counter(H : Hits) RETURN integer;

--SET and GET TARGET_SPEED PROCEDURE and FUNCTIONS*************

--**********************************************************************
--| Procedure: Set_Hit_Target_Speed
--| Input: Target Speed, Hit_Type
--| Output: Hit_Type
--**********************************************************************
PROCEDURE Set_Hit_Target_Speed(Spd : IN speeds.Speed; H : IN OUT Hits);

FUNCTION Get_Hit_Target_Speed(H : Hits) RETURN Realnum.Real;

PROCEDURE Calc_Hit_Target_Angle(H : IN OUT Hits);

PROCEDURE Set_Hit_Target_Angle;
--| Input: Target Angle, Hit_Type
--| Output: Hit_Type
--| Description: Sets Target Angle into Hit_Type to Inputed Value

**********************************************************************
PROCEDURE Set_Hit_Target_Angle(Target_Angle : IN degrees.Degree;
        H : IN OUT Hits);

**********************************************************************
--| Function: Get_Hit_Target_Angle
--| Input: Hit_Type
--| Output: Target Angle(degrees.Degree)
--| Description: Returns Hit Target Angle Value of inputed Hit_Type

FUNCTION Get_Hit_Target_Angle(H : Hits) RETURN degrees.Degree;

CALC_HIT_LAT_LONG

--| Procedure: Calc_Hit_Lat_Long
--| Input: Hit_Type
--| Output: None
--| Description: Calculates Latitude and Longitude of the inputed Hit based on
--| Bearing and Range and OwnShips Lat/Long. This Procedures calls
--| lat_long.Calc_Lat_Long procedure to calculates the new lat/long.

71
PROCEDURE Calc_Hit_Lat_Long (Lat1 : IN lat_long.Latitude;
    Long1 : IN lat_long.Longitude; H : IN OUT Hits);

PRIVATE

TYPE Hits IS RECORD

    Bear: degrees.Degree := 0.0;
    Rng : Realnum.Real := 0.0; --Stored in Yards
    Lat : lat_long.Latitude;
    Lon : lat_long.Longitude;
    Dat : dates.Date_Type := dates.Get_Date_From_System;
    Time: times.Time_Type := times.Get_Time_Of_Day;
    Target_Cse : degrees.Degree := 0.0;
    OwnShip_Cse : degrees.Degree := 0.0;
    Target_Speed : speeds.Speed := 0.0;
    OwnShip_Speed : speeds.Speed := 0.0;
    Target_Angle : degrees.Degree := 0.0;
    Hit_Counter : integer := 0;

END RECORD;

END hit;
WITH Ada.Numerics.Generic_Elementary_Functions;

WITH Ada.Numerics; USE Ada.Numerics;

PACKAGE Realnum IS

TYPE Real IS DIGITS 12;

PACKAGE Real_Functions IS

  NEW Generic_Elementary_Functions(Real);

END Realnum;

A-5 SPEEDS.ADS

--****************************************************************

--| File : speeds.ads

--| Purpose : This file defines a type Speed that is of Realnum.Real with digits

--| 1 precision and a range of 0.0 to 1000.0 knots
WITH Realnum; USE Realnum;

PACKAGE speeds IS

SUBTYPE Speed IS Real DIGITS 1 RANGE 0.0..1000.0;

-- Max speed set to 1000.0 Knots;
-- this is designed this way because our program is designed to track
-- surface/subsurface contacts, and possibly helicopters. This upper range is
-- set because a user could input a wrong bearing that causes the speed
-- calculated to go outside normal reasonable knots.

END speeds;

A-6 DEGREES.ADS

--*******************************************************************************

--| File : degrees.ads
--| Purpose : This file defines a type Degree that is of Realnum.Real type with
--| a precision of digits one and range of 0.0 to 359.9
--*******************************************************************************
WITH Realnum; USE Realnum;

PACKAGE degrees IS

SUBTYPE Degree IS Real DIGITS 1 RANGE 0.0..359.9;

END degrees;

A-7 OWNSHIP.ADS

--**********************************************************************
--| FILE: ownship.ads
--| AUTHOR: Joey L. Frantzen, Naval Post Graduate School
--| LAST MODIFIED: 11 September 2001
--| OPERATING ENVIRONMENT: Windows 2000(Designed to be O/S Independent)
--| COMPILER: GNAT 3.13p
--| DESCRIPTION: This class is contains all of the data types and
--| functionality for your ownship.
--| INPUTS: N/A
--| OUTPUTS: Outputs based on what Functions are requested and run.
--| Process: None to note
--| Assumptions: N/A
--| Warnings: None
--**********************************************************************

WITH lat_long;

WITH degrees;
WITH speeds;
WITH Utilities;
WITH file_io;
WITH historical_io;
WITH times;
WITH Ada.Strings.Fixed;

PACKAGE ownship is

--******************************************************************************
--| Procedure: Init_OwnShip
--| Input:  OS Number, OS Id, OS Course, OS Speed, Latitude, Longitude
--| Output: None
--| Description: Initializes and sets all inputed values into Ownship record
--******************************************************************************

PROCEDURE Init_OwnShip(OS_Number : IN integer; OS_Id : IN String;
    OS_Cse : IN degrees.Degree; OS_Speed : IN speeds.Speed;
    Lat1 : IN lat_long.Latitude; Long1 : IN lat_long.Longitude);

    --Eventually LAT/LONG will be pulled from either Manual/GPS entry
    --For now will pass in manually

--******************************************************************************
--| Function: Reset_OwnShip

76
--| Input: None
--| Output: Boolean

--| Description: Resets all inputed values into Ownship record(Zero's Out)

FUNCTION Reset_Ownship RETURN Boolean;

--| Procedure: Set_OwnShip_Number

--| Input: Integer
--| Output: None

--| Description: Sets Ownship Number into Ownship record Ownship_Number

PROCEDURE Set_OwnShip_Number(OS_Number : IN integer);

--| Function: Get_OwnShip_Number

--| Input: None
--| Output: Integer

--| Description: Gets Ownship Number info from Ownship record Ownship_Number

FUNCTION Get_OwnShip_Number RETURN integer;

--| Procedure: Set_OwnShip_Id
| Input:  String |
| Output: None |
| Description: Sets Ownship Number into Ownship record Ownship_Id |

PROCEDURE Set_OwnShip_Id(OS_Id : IN String);

| Function: Get_OwnShip_Id |
| Input: None |
| Output: String |
| Description: Gets Ownship Id info from Ownship record Ownship_Id |

FUNCTION Get_OwnShip_Id RETURN string;

| Procedure: Set_OwnShip_Cse |
| Input: Course (degrees.degree) |
| Output: None |
| Description: Sets Ownship Course into Ownship record Ownship_Cse |

PROCEDURE Set_OwnShip_Cse(OS_Cse : IN degrees.Degree);

| Function: Get_OwnShip_Cse |
--| Input: None
--| Output: degrees.Degree(Course)
--| Description: Gets Ownship Course info from Ownship record Ownship_Cse

FUNCTION Get_OwnShip_Cse RETURN degrees.Degree;

--| Procedure: Set_OwnShip_Speed
--| Input: Speed (speeds.Speed)
--| Output: None
--| Description: Sets Ownship Speed into Ownship record Ownship_Speed

PROCEDURE Set_OwnShip_Speed(OS_Speed : IN speeds.Speed);

--| Function: Get_OwnShip_Speed
--| Input: None
--| Output: speeds.Speed(Speed)
--| Description: Gets Ownship Speed info from Ownship record Ownship_Speed

FUNCTION Get_OwnShip_Speed RETURN speeds.Speed;

--| Procedure: Set_OwnShip_Lat

---
--| Input:  Degrees, Minutes, Seconds, Hemisphere Sign
--| Output: None
--| Description: Sets Ownship Latitude into Ownship record Ownship_Lat

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

PROCEDURE Set_OwnShip_Lat(D : IN lat_long.Lat_Degree; M: IN lat_long.Min_utes;
                          S : IN lat_long.Sec_onds; Sign : IN Character);

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

--| Function: Get_OwnShip_Lat
--| Input:  None
--| Output: lat_long.Latitude
--| Description: Gets Ownship Latitude info from Ownship record Ownship_Lat

FUNCTION Get_OwnShip_Lat RETURN lat_long.Latitude;

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

--| Procedure: Set_OwnShip_Long
--| Input:  Degrees, Minutes, Seconds, Hemisphere Sign
--| Output: None
--| Description: Sets Ownship Longitude into Ownship record Ownship_Lon

PROCEDURE Set_OwnShip_Long(D : IN lat_long.Long_Degree;
                           M : IN lat_long.Min_utes;
                           S : IN lat_long.Sec_onds; Sign : IN Character);
FUNCTION Get_OwnShip_Long RETURN lat_long.Longitude;

PROCEDURE Load_Ownship_Data(Ownship_Data : IN file_io.OS_Data_Type);

TYPE own_ship IS RECORD
    OwnShip_Number : integer := 0;  --Set to Hull Number/Associated Number
    OwnShip_Ident : String(1..20) := Utilities.Get_Default_String;
        --max length is 20 characters
    OwnShip_Cse : degrees.Degree := 0.0;
    OwnShip_Speed : speeds.Speed := 0.0;  --Expressed in Knots
    Lat : lat_long.Latitude;

    --| Description: Sets Ownship Data Type info into Ownship record

PROCEDURE Load_Ownship_Data(Ownship_Data : IN file_io.OS_Data_Type);
Lon : lat_long.Longitude;

END RECORD;

OS_Data : file_io.OS_Data_Type;

File_Saved : Boolean;

OWNSHIP_CONST : CONSTANT integer := 1;

Ship_Data : historical_io.Historical_Data_Type(OWNSHIP_CONST);

Own_Ship_Info : own_ship;

END ownship;

A-8 TRACKS.ADS

--**********************************************************************
--| FILE: tracks.ads
--| AUTHOR: Joey L. Frantzen, Naval Post Graduate School
--| LAST MODIFIED: 14 September 2001
--| OPERATING ENVIRONMENT: Windows 2000(Designed to be O/S Independent)
--| COMPILER: GNAT 3.13p
--| DESCRIPTION:  This class is contains all of the data types and
--| functionality for a specific track and an array of tracks that contains a
--| link list of hits associated with that track.
--| INPUTS:  N/A
--| OUTPUTS: Outputs based on what Functions are requested and run.
--| Process: None to note
--| Assumptions: N/A
--| Warnings: None
WITH hit;
WITH times;
WITH dates;
WITH Realnum;
WITH degrees;
WITH speeds;
WITH ownship;
WITH lat_long;
WITH Utilities;
WITH file_io;
WITH historical_io;
WITH Ada.Strings;
WITH Ada.Strings.Fixed;
WITH Ada.Unchecked_Deallocation;
WITH Gdk.Color;

PACKAGE tracks IS

Number_Of_Tracks : integer := Utilities.LAST_TRACK_SLOT -
Utilities.BEGIN_TRACK_SLOT;
SUBTYPE Active_Index IS integer RANGE 1..Number_Of_Tracks;

--Maintains sorted List of all tracks that are active

TYPE Active_Track_List_Array IS ARRAY (Active_Index) OF integer;

TYPE List_Hit_Type; --Declaration of List_Hit_Type is promise for full
--declaration in follow on code.

TYPE List_Hit_Pointer_Type IS ACCESS List_Hit_Type; --Declares Type pointer to
--List_Hit_Type

TYPE List_Hit_Type IS

    RECORD
        H : hit.Hits;
        Next : List_Hit_Pointer_Type := NULL;
        Prev : List_Hit_Pointer_Type := NULL;
    END RECORD;

PROCEDURE Clear_Active_Track_List;

--| Procedure: Clear_Active_Track_List
--| Input: None
--| Output: None
--| Description: Sets all values in Active_Track_List to Zero.

PROCEDURE Clear_Active_Track_List;
FUNCTION Get_Active_Track_List RETURN Active_Track_List_Array;

PROCEDURE Get_A_New_Track_Number(Track_Num : OUT Utilities.Track_Number_Range;
                                      Can_Make_Track : OUT Boolean);
--| Procedure: Add_Track

--| Input: Track_Number, Track_Id, Bearing, Range, Track Scale

--| Output: None

--| Description: Checks if Track_Number IN is Available, if Available then
stores inputed data into Track Number slot of Track List Array. If Track
Num in is not available, algorithm finds a available then saves data into
next available track slot. Add_Track also calls Add_A_Hit, which builds a
new Hit with the bearing and range entered and puts it at the Head of
Hit_List. Additionally Saves Hit and Track Data into the Lastest and
Historical datafiles.

PROCEDURE Add_Track(Track_Num : IN Utilities.Track_Number_Range;
Track_Id : IN String;
Bear : IN degrees.Degree; Rng : IN Realnum.Real;
T_Scale : IN integer; Track_Color : Gdk.Color.Gdk_Color);

--| Procedure: Add_Track_Hit

--| Input: Track_Number, Bearing, Range, Track Scale

--| Output: None

--| Description: Stores Track Scale into Track_Num and Adds a New Hit to
Hit_List for associated Track Number via Add_A_Hit Procedure. Additionally
Saves Hit and Track Data into the Lastest and Historical datafiles.
PROCEDURE Add_Track_Hit(Track_Num : IN Utilities.Track_Number_Range;
    Bear : IN degrees.Degree;
    Rng : IN Realnum.Real; T_Scale : IN integer);

--*****************************************************************************

PROCEDURE Set_Purge_Time(New_Time : IN integer);

--*****************************************************************************

PROCEDURE Purge_Old_Tracks;

--*****************************************************************************

--| Procedure: Purge_Old_Tracks
--| Input: None
--| Output: None - At least, no return values
--| Description: Cycles through the Track_List Array and checks to see last hit
--| update for each track. If the last hit on that track is older then the set
--| Oldest Time then Purges this track from the array and resets the track data
PROCEDURE Purge_Old_Tracks;

--| Procedure: Change_Track_Number

--| Input: Request Track Number, Current Track Number

--| Output: Boolean representing if Requested Track Number was free.

--| Description: Algorithm has two cases. Case 1, the requested track number is

--| free and then current Track number is moved into requested track number

--| slot. Current Slot is then cleared. Track Free is Returned TRUE. Case 2,

--| Requested Track Number is not free. In this case, Track Info in Requested

--| Slot moved into first available slot. Then Current Track Number is moved

--| into wanted slot, and current slot is cleared. Track Free is returned True.

--| If in both cases there are NO free slots, Purge_Old_Tracks is executed and

--| both cases are tried again. If again, there are no free slots, then Track

--| Free is returned false. Historical and Latest File Database are updated

--| after each change.

PROCEDURE Change_Track_Number(Wanted_Track_Number : IN

Utilities.Track_Number_Range;

Current_Track_Number : IN

Utilities.Track_Number_Range;
---| Procedure: Set_Track_Cse_And_Speed
---| Input: Track Number, Course, Speed
---| Output: None

---| Description: Saves Track Course and Speed into the Last Hit Target_Course
---| and Speed variables, then Calls Calculate Target Angle in the Hit Class,
---| Saves the Hit into the latest and historical databases. Then saves course
---| and speed in Track and saves target angle in track. Then the algorithm
---| saves the updated Track in Latest and historical databases.

---|******************************************************************************
---**| Procedure: Set_Track_Id
---| Input: Track Number, String(Track Id)
---| Output: None

---| Description: Saves Track Id into the Track Number variable called Track_Id.
---| Saves the New Track Information into the latest and historical database
---| files.
PROCEDURE Set_Track_Id(Track_Num : IN Utilities.Track_Number_Range;
                               Track_Id : IN String);

PROCEDURE Set_Track_Scale(Track_Num : IN Utilities.Track_Number_Range;
                           Scale : IN integer);

PROCEDURE Set_Track_Color(Track_Num : IN Utilities.Track_Number_Range;
                           Track_Color : IN Gdk.Color);

-- Procedure: Set_Track_Scale
-- Input:  Track Number, Scale(Integer)
-- Output: None
-- Description: Saves Track Scale into the Track Number variable called
-- Track_Scale. Algorithm Checks if Track Scale has been set, if not then
-- saves inputed Track Scale If Track Scale already set, then does a
-- comparison and saves the Track Scale whose value is greater. I.E. If I go
-- from a 5:1 to 10:1 Scale then Track Scale is 10:1 overall. Saves the Track
-- Information into the latest and historical database files.

PROCEDURE Set_Track_Scale(Track_Num : IN Utilities.Track_Number_Range;
                           Scale : IN integer);

-- Procedure: Set_Track_Color
-- Input:  Track Number, Track_Color(Gdk.Color)
-- Output: None
-- Description: Saves Track Color into the Track Number variable called
PROCEDURE Set_Track_Color(Track_Num : IN Utilities.Track_Number_Range;
    Track_Color : IN Gdk.Color.Gdk_Color);

PROCEDURE Set_Track_CPA(Track_Num : IN Utilities.Track_Number_Range;
    Bearing : IN Utilities.My_Degree;
    T_ime : IN integer; Rng : IN Realnum.Real);

PROCEDURE Set_Track_CPA_Bearing
    Bearing : IN Utilities.My_Degree;
    T_ime : IN integer; Rng : IN Realnum.Real);
--| Description: Saves CPA Bearing into the associated Track Number CPA_Bearing
--| variable. Saves the New Track Information into the latest and historical
--| database files.

--**********************************************************************

PROCEDURE Set_Track_CPA_Bearing(Track_Num : IN

Utilities.Track_Number_Range;

Bearing : IN Utilities.My_Degree);

--**********************************************************************

--| Procedure: Set_Track_CPA_Time
--| Input:  Track Number, Time
--| Output: None
--| Description: Saves CPA Time into the associated Track Number CPA_Time
--| variable. Saves the New Track Information into the latest and historical
--| database files.

--**********************************************************************

PROCEDURE Set_Track_CPA_Time(Track_Num : IN Utilities.Track_Number_Range;

T ime : IN integer);

--**********************************************************************

--| Procedure: Set_Track_CPA_Range
--| Input:  Track Number, Range
--| Output: None
--| Description: Saves CPA Range into the associated Track Number CPA_Range
PROCEDURE Set_Track_CPA_Range(Track_Num : IN Utilities.Track_Number_Range; 
    Rng : IN Realnum.Real);

PROCEDURE Set_Track_CPA_Faired(Track_Num : IN Utilities.Track_Number_Range; 
    Fair : IN boolean);

PROCEDURE Set_Track_CPA_Faired(Track_Num : IN Utilities.Track_Number_Range; 
    Fair : IN boolean);

FUNCTION Get_Purge_Time

Input: None
Output: integer (represents Purge time in hours)
Description: Returns Purge_Time variable which represents time late in
hours that Purge_Tracks drops tracks that last hit was greater than or
equal to Purge_Time.

FUNCTION Get_Purge_Time RETURN integer;

FUNCTION Get_Track_Cse(Track_Num : IN Utilities.Track_Number_Range)
        RETURN degrees.Degree;

FUNCTION Get_Track_Speed(Track_Num : IN Utilities.Track_Number_Range)
        RETURN Realnum.Real;

FUNCTION Get_Track_Id

94
--| Input: Track Number
--| Output: String(represents Track Id in String)
--| Description: Returns Track Id associated with Track Number entered.

FUNCTION Get_Track_Id(Track_Num : IN Utilities.Track_Number_Range) RETURN String;

--| Function: Get_Track_Scale
--| Input: Track Number
--| Output: integer(represents Track Scale in integer)
--| Description: Returns Track Scale associated with Track Number entered.

FUNCTION Get_Track_Scale(Track_Num : IN Utilities.Track_Number_Range) RETURN integer;

--| Function: Get_Track_Color
--| Input: Track Number
--| Output: Gdk.Color.Gdk_Color(represents Track Color)
--| Description: Returns Track Color associated with Track Number entered.

FUNCTION Get_Track_Color(Track_Num : IN Utilities.Track_Number_Range) RETURN Gdk.Color.Gdk_Color;
--| Function: Get_Track_Last_Hit_Lat

--| Input: Track Number

--| Output: lat_long.Latitude(representing Tracks Last Latitude Coordinate)

--| Description: Returns Track Latitude(of last hit) associated with Track Number entered.

FUNCTION Get_Track_Last_Hit_Lat(Track_Num : IN Utilities.Track_Number_Range) RETURN lat_long.Latitude;

--| Function: Get_Track_Last_Hit_Long

--| Input: Track Number

--| Output: lat_long.Longitude(Represents Tracks Last Longitude Coordinate)

--| Description: Returns Track Longitude(of last hit) associated with Track Number entered.

FUNCTION Get_Track_Last_Hit_Long(Track_Num : IN Utilities.Track_Number_Range) RETURN lat_long.Longitude;
|-- Output: Utilities.My_Degree(represents Track CPA in Degrees)
|-- Description: Returns Track CPA Bearing associated with Track Number entered.

FUNCTION Get_Track_CPA_Bearing(Track_Num : IN Utilities.Track_Number_Range) RETURN Utilities.My_Degree;

|-- Function: Get_Track_CPA_Time
|-- Input: Track Number
|-- Output: Integer(represents Track CPA in Time(1230)
|-- Description: Returns Track CPA Time associated with Track Number entered.

FUNCTION Get_Track_CPA_Time(Track_Num : IN Utilities.Track_Number_Range) RETURN integer;

|-- Function: Get_Track_CPA_Range
|-- Input: Track Number
|-- Output: Realnum.Real(represents Track CPA in Range
|-- Description: Returns Track CPA Range associated with Track Number entered.

FUNCTION Get_Track_CPA_Range(Track_Num : IN Utilities.Track_Number_Range) RETURN Realnum.Real;
FUNCTION Get_Track_CPA_Faired(Track_Num : IN Utilities.Track_Number_Range) RETURN boolean;

PROCEDURE Get_CPA_Info(Track_Num : IN Utilities.Track_Number_Range;
    First_Bearing : IN OUT degrees.Degree;
    Second_Bearing : IN OUT degrees.Degree;
    First_Range : IN OUT Realnum.Real;
    Second_Range : IN OUT Realnum.Real;
    First_Time : IN OUT times.Time_Type;
    Second_Time : IN OUT times.Time_Type;
--| Procedure: Get_Hit_Counter

|--| Input: Track Number

|--| Output: Hit_Counter

|--| Description: Returns Hit Counter, which represents number of Hits a track
|--| has entered.

FUNCTION Get_Hit_Counter(Track_Num : IN Utilities.Track_Number_Range)

    RETURN integer;

--| Procedure: Recalc_All_Track_CPAs

|--| Input: None

|--| Output: None

|--| Description: This Procedure has not been implemented yet.

PROCEDURE Recalc_All_Track_CPAs;

--| Procedure: Get_Faired_Track_Info

|--| Input: Track Number

|--| Output: First Bearing, Last Bearing, First Range, Last Range, First Time,
--| Last Time. Description: Receives a Track number and gets the First and Last
--| Hit and returns the Bearing, Range, and Time of each of the hits.

--**********************************************************************
PROCEDURE Get_Faired_Track_Info(Track_Num : IN
  Utilities.Track_Number_Range;
    Bearing1 : OUT Utilities.My_Degree;
    Bearing2 : OUT Utilities.My_Degree;
    Range1 : OUT Realnum.Real;
    Range2 : OUT Realnum.Real;
    Time1 : OUT times.Time_Type;
    Time2 : OUT times.Time_Type);
--**********************************************************************

--| Procedure: Get_Last_Track_Bearing_Range
--| Input: Track Number
--| Output: Bearing, Range of Track Number
--| Description: Procedure Returns the values of Last Bearing and Range so that
--| Moboard can load values into the list.

--**********************************************************************
PROCEDURE Get_Last_Track_Bearing_Range(Track_Num : IN
  Utilities.Track_Number_Range;
    Bearing : OUT Utilities.My_Degree;
    Rng : OUT Realnum.Real);
--**********************************************************************
--| Procedure: Get_Track_Info
--| Input: Track Number
--| Output: Bearing, Range, Course, Speed, and Target Angle of Track Number
--| Description: Procedure Returns the values of Bearing, Range, Course, Speed, and Target Angle of specified track number so Moboard can load values into
--| the list.

--**********************************************************************
PROCEDURE Get_Track_Info(Track_Num : IN Utilities.Track_Number_Range;
    Brg : OUT Utilities.My_Degree; Rng : OUT Realnum.Real;
    Cse : OUT Utilities.My_Degree;
    Speed : OUT speeds.Speed;
    Target_Angle : OUT Utilities.My_Degree);

--**********************************************************************

--| Function: Drop_Track
--| Input: Track Number
--| Output: Boolean(represents whether track was successfully dropped or not
--| Description: Calls Clear_Track_Data which resets and clears all track and
--| associated Hit data. Also removes track and associated Hits from Latest
--| Database File. Checks if Track Number is equal to Closest_CPA_Track_Number
--| then recalculates it.

--**********************************************************************

FUNCTION Drop_Track(Track_Num : IN Utilities.Track_Number_Range) RETURN
    Boolean;
FUNCTION Command_Data_Reset RETURN Boolean;

FUNCTION Load_Last_Saved_Data RETURN Boolean;

FUNCTION Get_Track_Hit_List RETURN List_Hit_Pointer_Type;
--| Description: Returns the Head Pointer of Hit_List for the associated Track

FUNCTION Get_Track_Hit_List(Track_Num : IN Utilities.Track_Number_Range)
    RETURN List_Hit.Pointer_Type;

--| Function: Get_Closest_CPA_Track_Number

FUNCTION Get_Closest_CPA_Track_Number RETURN
    Utilities.Track_Number_Range;

--| Function: Get_Number_Of_Active_Tracks

FUNCTION Get_Number_Of_Active_Tracks RETURN integer;

PRIVATE
TYPE Track_Type IS RECORD

    Track_Number : Utilities.Track_Number_Range := Utilities.LAST_TRACK_SLOT;
    -- Initialize All Records Track_Number

    Track_Id : String(1..20) := Utilities.Get_Default_String;
    -- max length is 20 characters

    Track_Cse : degrees.Degree := 0.0;       -- average/faired Course of Track

    Track_Speed : Realnum.Real := 0.0;       -- average/faired Speed of Track

    Target_Angle : degrees.Degree := 0.0;

    Track_Scale : integer := 0;

    -- have to change later to right type

    CPA_Bearing : Utilities.My_Degree := 0;

    CPA_Range : Realnum.Real := 0.0;

    CPA_Time : integer := 0;

    CPA_Is_Faired : boolean := FALSE;
    -- bit setting on whether CPA Values are faired values

    Hit_List : List_Hit_Pointer_Type;

    Hit_Counter : integer := 0;

END RECORD;

-- Defines the Upper and Lower Limits for Purge Time settings

BOTTOM_OLD_TIME : CONSTANT integer := 1;

TOP_OLD_TIME : CONSTANT integer := 12;
--Represents the Time in hours when Purge_Tracks will delete any tracks
--older then this set time. Default is 12 hours
Purge_Time : integer RANGE BOTTOM_OLD_TIME..TOP_OLD_TIME := 12;

TYPE Track_Array IS ARRAY (Utilities.Track_Number_Range) OF Track_Type;
--Holds records of Track_Type and maintains all info on Tracks
Track_List : Track_Array;

--Maintains sorted List of all tracks that are active
Active_Track_List : Active_Track_List_Array := ( others => 0 );

--Maintains total number of active tracks
Number_Of_Active_Tracks : integer := 0;

--Variable holds temporary Track data for saving to File Database
Track_Data : file_io.Track_Data_Type;

--Variable holds temporary Hit data for saving to File Database
Hit_Data : file_io.Hit_Data_Type;

--Constant sets the Historical Database Type to contain Track Info
TRACK_CONST : CONSTANT integer := 2;
-- Constant sets the Historical Database Type to contain Hit Info

HIT_CONST : CONSTANT integer := 3;

-- Variable holds temporary Track data for saving to Historical Database

Track_Hist_Data : historical_io.Historical_Data_Type(TRACK_CONST);

-- Variable holds temporary Hit data for saving to Historical Database

Hit_Hist_Data : historical_io.Historical_Data_Type(HIT_CONST);

Track_Saved : Boolean;
Hit_Saved : Boolean;

-- Represents the track number that has the
-- current closest CPA value in yards.

Closest_CPA_Track_Number : Utilities.Track_Number_Range :=
Utilities.LAST_TRACK_SLOT;

END tracks;

A-9 FILE_IO.ADS

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

--FILE: file_io.ads

--AUTHOR: Joey L. Frantzen, Naval Post Graduate School

--LAST MODIFIED: 12 September 2001
--| OPERATING ENVIRONMENT: Windows 2000(Designed to be O/S Independent)
--| COMPILER: GNAT 3.13p
--| DESCRIPTION: This class is contains all of the data types and
--| functionality allowing to save a database for crash recovery, restart
--| recovery, or error recovery from a database that holds all current
--| information.
--| INPUTS: Based on specific functions and procedures are called.
--| OUTPUTS: Based on what functions and procedures are called.
--| Process: None to note
--| Assumptions: N/A
--| Warnings: None

--**********************************************************************

WITH utilities;
WITH speeds;
WITH degrees;
WITH Realnum;
WITH lat_long;
WITH times;
WITH dates;
WITH GNAT.IO_Aux;
WITH Gdk.Color;
WITH Ada.Direct_IO;
PACKAGE file_io IS

-- record type represents all pertinent data for ownship and is used in file io operations

TYPE OS_Data_Type IS RECORD
    OwnShip_Number : integer := 99; -- Set to Hull Number/Associated Number
    OwnShip_Ident : String(1..20) := Utilities.Get_Default_String;
        -- max length is 20 characters
    OwnShip_Cse : degrees.Degree := 0.0;
    OwnShip_Speed : speeds.Speed := 0.0; -- Expressed in Knots
    Lat : lat_long.Latitude;
    Lon : lat_long.Longitude;
END RECORD;

-- record type represents all pertinent data for Hit type and is used in file io operations

TYPE Hit_Data_Type IS RECORD
    Track_Number : Utilities.Track_Number_Range := Utilities.LAST_TRACK_SLOT;
    Bear : degrees.Degree := 0.0;
        Rng : Realnum.Real := 0.0; -- Stored in Yards
    Lat : lat_long.Latitude;
    Lon : lat_long.Longitude;
END RECORD;
Dat : dates.Date_Type;

T_ime: times.Time_Type;

Target_Cse : degrees.Degree := 0.0;

OwnShip_Cse : degrees.Degree := 0.0;

Target_Speed : speeds.Speed := 0.0;

OwnShip_Speed : speeds.Speed := 0.0;

Target_Angle : degrees.Degree := 0.0;

Hit_Counter : integer := 0;

END RECORD;

--record represents all pertinent data for track type and is used in file io

--operations

TYPE Track_Data_Type IS

RECORD

  Track_Number : Utilities.Track_Number_Range := Utilities.LAST_TRACK_SLOT;

    --Initialize All Records Track_Number

  Track_Id : String(1..20) := Utilities.Get_Default_String; --max length is

       --20 characters

  Track_Cse : degrees.Degree := 0.0; -- Course of Track

  Track_Speed : Realnum.Real := 0.0; -- Speed of Track

  Target_Angle : degrees.Degree := 0.0;

  Track_Scale : integer := 0;
CPA_Bearing: Utilities.My_Degree := 0;
CPA_Range: Realnum.Real := 0.0;
CPA_Time: integer := 0;
CPA_Is_Faired: boolean := FALSE;
Hit_Counter: integer := 0;

END RECORD;

FUNCTION Save_Last_OwnShip_State(Ownship_Data: IN OS_Data_Type) RETURN Boolean;

-- Function: Save_Last_Track_State
-- Input: Track_Date Type
-- Output: Boolean
-- Description: This is a Latest(Last State) recovery file. Saves the Last
-- updated information about ownship into file called ownship_data.db If the
-- file does not exist then the algorithm creates the file and saves the data.

FUNCTION Save_Last_Track_State(Track_Data: IN Track_Type) RETURN Boolean;

-- Procedure: Save_Last_Track_State
-- Input: Track_Date Type
-- Output: Boolean
-- Description: This is a Latest(Last State) recovery file. Saves the Last

110
PROCEDURE Save_Last_Track_State(Track_Data : IN OUT Track_Data_Type;
                          Track_Saved : OUT Boolean);

PROCEDURE Save_Last_Hit_State(Hit_Data : IN OUT Hit_Data_Type;
                                 Hit_Saved : OUT Boolean);

PROCEDURE Save_Last_Hit_State(Hit_Data : IN OUT Hit_Data_Type;
                                 Hit_Saved : OUT Boolean);

FUNCTION Save_Two_Hit_Track_Numbers
    --| Input: Wanted Track Number, Current Track Number
    --| Output: Boolean
    --| Description: This procedure takes two track numbers, one that the user
    --| wants to change the track number two, and the previous used track number.
    --| This procedure cycles through the Hit database and changes the hits
FUNCTION Swap_Two_Hit_Track_Numbers(Wanted : IN Utilities.Track_Number_Range;
Current : IN Utilities.Track_Number_Range)
RETURN Boolean;

FUNCTION Swap_Three_Hit_Track_Numbers(Wanted : IN Utilities.Track_Number_Range;
Current : IN Utilities.Track_Number_Range;
New_Track : IN Utilities.Track_Number_Range)
RETURN Boolean;

FUNCTION Drop_Track_From_Track_State
--| with the current track track number to associate with the wanted track
--| number.
--**********************************************************************

--| Function: Save_Three_Hit_Track_Numbers
--| Input: Wanted Track Number, Current Track Number, New Track Number
--| Output: Boolean
--| Description: This procedure takes three track numbers, one that the user
--| wants to change the track number too, and the previous used track number,
--| and the new track number. This procedure cycles through the Hit database
--| and changes the the hits to the associatede track numbers.
--**********************************************************************

FUNCTION Save_Three_Hit_Track_Numbers(Wanted : IN Utilities.Track_Number_Range;
Current : IN Utilities.Track_Number_Range;
New_Track : IN Utilities.Track_Number_Range)
RETURN Boolean;
--| Input: Track Number
--| Output: Boolean
--| Description: This procedure deletes inputed track number from the
--| Latest(Last State). This is done by overwriting the index number(track
--| number) position with a null value.

FUNCTION Drop_Track_From_Track_State(Track_Num : IN
    Utilities.Track_Number_Range) RETURN Boolean;

-- Function: Drop_All_Track_Hits_From_Hit_State
--| Input: Track Number
--| Output: Boolean
--| Description: This procedure deletes all hits associated with inputed track
--| number from the Latest(Last State). This is done by overwriting the hits
--| associated with index number(track number) with a null value.

FUNCTION Drop_All_Track_Hits_From_Hit_State(Track_Num : IN
    Utilities.Track_Number_Range) RETURN Boolean;

-- Function: Delete_OS_Latest_Files
--| Input: None
--| Output: Boolean
Description: This procedure deletes all Latest files Ownship_database.db, track_database.db, and hit_database.db.

FUNCTION Delete_Latest_Files RETURN Boolean;

FUNCTION Get_Ownship_Data_From_File RETURN OS_Data_Type;

FUNCTION Get_All_Saved_Tracks RETURN Track_Data_Type, Index, and Boolean;

Description: This procedure returns the Track_Data_Type stored in Track database. The procedure will cycle through the complete database track by track as each sequential call is made. Once all tracks are loaded the Loaded boolean is returned with value True.
PROCEDURE Get_All_Saved_Tracks(Track_Data : OUT Track_Data_Type;
   Index : IN OUT Utilities.Track_Number_Range;
   Loaded : OUT Boolean; File_Exists : OUT Boolean);

PROCEDURE Get_All_Saved_Hits(Hit_Data : OUT Hit_Data_Type;
   Index : IN OUT integer;
   Loaded : OUT Boolean);

END file_io;

A-10 HISTORICAL_IO.ADS

PROCEDURE Get_All_Saved_Hits(Hit_Data : OUT Hit_Data_Type;
   Index : IN OUT integer;
   Loaded : OUT Boolean);

END file_io;
LAST MODIFIED: 12 September 2001

OPERATING ENVIRONMENT: Windows 2000(Designed to be O/S Independent)

COMPILER: GNAT 3.13p

DESCRIPTION: This class is contains all of the data types and functionality allowing to save a database for historical playback or display, this is a historical database by day, each new day there is a new file created or auto input.

Due to Ada's limited ability to do file io, delete, etc. There is one date type that exists based upon what the user case is, ownship data, track data, or hit data. There is also a Sentry case, that is saved into the historical file prior to each historical save, allowing the customer to read the sentry, determine the data structre (next) and then reading the data structure, etc.

INPUTS: Based on specific functions and procedures are called.

OUTPUTS: Based on what functions and procedures are called.

Process: None to note

Assumptions: N/A

Warnings: None

**********************************************************************
WITH utilities;
WITH speeds;
WITH degrees;
WITH Realnum;
WITH lat_long;
WITH times;
WITH dates;
WITH GNAT.IO_Aux;
WITH Gdk.Color;
WITH Ada.Sequential_IO;

PACKAGE historical_io IS

TYPE Historical_Data_Type(In_Use : integer := 1) IS

RECORD

CASE In_Use IS

WHEN 1 =>

Time_Stamp : times.Time_Type := times.Get_Time_Of_Day;
OwnShip_Number : integer := 99; -- Set to Hull Number/Associated Number
OwnShip_Ident : String(1..20) := Utilities.Get_Default_String; -- max
length is 20 character
OwnShip_Cse : degrees.Degree := 0.0;
OwnShip_Speed : speeds.Speed := 0.0; -- Expressed in Knots
OwnShip_Lat : lat_long.Latitude;
OwnShip_Lon : lat_long.Longitude;

WHEN 2 =>

Track_Number : Utilities.Track_Number_Range :=

    Utilities.LAST_TRACK_SLOT; -- Initialize All Records Track_Number
Track_Id : String(1..20) := Utilities.Get_Default_String; -- max length is
20 characters
Track_Cse : degrees.Degree := 0.0; -- Course of Track
Track_Speed : Realnum.Real := 0.0; -- Speed of Track
Track_Target_Angle : degrees.Degree := 0.0;
Track_Scale : integer := 0;
CPA_Bearing : Utilities.My_Degree := 0;
CPA_Range : Realnum.Real := 0.0;
CPA_Time : integer := 0;
CPA_Is_Faired : boolean := FALSE;
Track_Hit.Counter : integer := 0;

WHEN 3 =>

Hit.Counter : integer := 0;

Hit_Track_Number : Utilities.Track_Number_Range :=

    Utilities.LAST_TRACK_SLOT;

Bear : degrees.Degree := 0.0;
Rng : Realnum.Real := 0.0; --Stored in Yards
Hit_Lat : lat_long.Latitude;
Hit_Lon : lat_long.Longitude;
Dat : dates.Date_Type;
Time : times.Time_Type;
Target_Cse : degrees.Degree := 0.0;
Hit_OwnShip_Cse : degrees.Degree := 0.0;
Target_Speed : speeds.Speed := 0.0;
Hit_OwnShip_Speed : speeds.Speed := 0.0;
Hit_Target_Angle : degrees.Degree := 0.0;

WHEN 4 =>

Sentry : integer := 0; --Great and Many thanks go out to Ken Ehresman.

WHEN OTHERS =>

NULL;

END CASE;

END RECORD;

--*********************************************************************
--| Procedure: New_Day_Change_File_Name
--| Input: None
--| Output: None
--| Description: This procedure retrieves the current system date, converts the
--| date into a string, for example 08/08/2001 is converted into 20010808.db
--| and this is saved into the Historical File Name variable.
--*********************************************************************

PROCEDURE New_Day_Change_File_Name;

--*********************************************************************
--| Function: Save_Data_To_File
--| Input: Historical Data Type
--| Output: Boolean
--| Description: This is a historical data file. Saves the historical data
information based on what case it is. Case 1, 2, 3 are data types for
ownership, tracks, and hits. The data is saved in a file based on the date
i.e. 20010808.db is for Aug 8 2001

FUNCTION Save_Data_To_File(Data_In : IN Historical_Data_Type;
Data_Case : IN integer) RETURN Boolean;

PRIVATE

--stores the current date based off the system clock
Current_Date : integer := 0;

--stores the current file name for the historical database
Historical_File_Name : String(1..12) := "2001hist.db ";

END historical_io;

A-11 LAT_LONG.ADS

--**********************************************************************
--FILE: lat_long.ads
--AUTHOR: Joey L. Frantzen, Naval Post Graduate School
--LAST MODIFIED: 11 September 2001
--OPERATING ENVIRONMENT: Windows 2000(Designed to be O/S Independent)
--COMPILER: GNAT 3.13p
--DESCRIPTION: This Class defines a Latitude and Longitude type and

120
contains all of the functions and procedures to calculate a new
latitude and longitude when given a point that is at a specific bearing
and range. This class also handles computing distance between two
given latitude and longitude coordinates. This class contains functions
to convert between Kilometers, Yards, and Nautical Miles. This Class
also contains two functions that call the Gps Class and retrieve a latitude
and longitude.
INPUTS: Ada.Numerics, Realnum.Real
OUTPUTS: Outputs based on user request and data input
Process: None to note
Assumptions: N/A
Warnings: None

WITH Ada.Numerics;
WITH Realnum; USE Realnum;
USE TYPE Realnum.Real;

PACKAGE lat_long IS

--Best Math Set Constant Declarations
== 3.14159_26535_89793_23846_26433_83279_50288_41971_69399_37511

TWO_PI: CONSTANT Realnum.Real := 2.0 * Ada.Numerics.Pi;  -- Two Pi

DEG_TO_RADIAN: CONSTANT Realnum.Real := Ada.Numerics.Pi / 180.0;  -- Deg to Radian

RADIANS_TO_DEG: CONSTANT Realnum.Real := 180.0 / Ada.Numerics.Pi;  -- Radian to Deg

FETH: CONSTANT Realnum.Real := 0.335_28106_6474E-2;  -- Flatting Constant (about 1/298)

RE: CONSTANT Realnum.Real := 6378.137;  -- Radius Earth at Equator = Semi-Major Axis

ESQ: CONSTANT Realnum.Real := 2.0 * FETH - (FETH * FETH);

A: CONSTANT Realnum.Real := 1000.0 * RE;

TOL: CONSTANT Realnum.Real := 5.0E-15;

SUBTYPE Min_utes IS integer RANGE 0..59;
SUBTYPE Sec_onds IS integer RANGE 0..59;
SUBTYPE Lat_Degree IS integer RANGE 0..90;
SUBTYPE Long_Degree IS integer RANGE 0..180;

TYPE Latitude IS private;
TYPE Longitude IS private;
--| Function: Get_NM_From_Yards(Realnum.Real)

--| Input: Yards

--| Output: Nautical Miles

--| Description: Algorithm converts Yards to Nautical Miles is taken from

--| pg. 7 of Piloting and Dead Reckoning 3rd Ed written by

--| H.H. Shufeldt, CAPT, USNR (Ret.).

--| This standard was accepted as Official by the United States in 1959.

FUNCTION Get_NM_From_Yards(Yard1 : Realnum.Real) RETURN Realnum.Real;

--| Function: Get_Yards_From_NM(Realnum.Real)

--| Input: Nautical Miles

--| Output: Yards

--| Description: Algorithm to convert Nautical Miles to Yards is taken from

--| pg. 7 of Piloting and Dead Reckoning 3rd Ed written by

--| H.H. Shufeldt, CAPT, USNR (Ret.). This standard was

--| This Standard was accepted as Official by the United States in 1959.

FUNCTION Get_Yards_From_NM(Nat1 : Realnum.Real) RETURN Realnum.Real;

--| Function: Get_NM_From_KM(Realnum.Real)

--| Input: Kilometers

123
--| Output: Nautical Miles;

--| Description: Algorithm to convert Kilometers to Nautical Miles is taken from

--| pg. 7 of Piloting and Dead Reckoning 3rd Ed written by

--| H.H. Shufeldt, CAPT, USNR (Ret.). This standard was

--| This Standard was accepted as Official by the United States in 1959.

FUNCTION Get_NM_From_KM(Kilo : Realnum.Real) RETURN Realnum.Real;

--| Function: Get_Yards_From_KM(Realnum.Real)

--| Input: Kilometers

--| Output: Yards

--| Discription: Algorithm to convert Kilometers to Yards is taken from

--| pg. 7 of Piloting and Dead Reckoning 3rd Ed written by

--| H.H. Shufeldt, CAPT, USNR (Ret.). This standard was

--| This Standard was accepted as Official by the United States in 1959.

FUNCTION Get_Yards_From_KM(Kilo : Realnum.Real) RETURN Realnum.Real;

--| Function: Get_KM_From_Yards(Realnum.Real)

--| Input: Yards

--| Output: Kilometers

--| Description: Algorithm to convert Yards to KM is taken from
FUNCTION Get_KM_From_Yards(Yard1 : Realnum.Real) RETURN Realnum.Real;

FUNCTION Get_KM_From_NM(NM1 : Realnum.Real) RETURN Realnum.Real;

FUNCTION Lat_To_Degrees(Lat1 : Latitude) RETURN Realnum.Real;

FUNCTION Lat_To_Degrees(Lat1 : Latitude) RETURN Realnum.Real;
FUNCTION Long_To_Degrees(Long1: Longitude) RETURN Realnum.Real;

FUNCTION Degrees_To_Lat(Deg: Realnum.Real) RETURN Latitude;

FUNCTION Degrees_To_Long(Deg: Realnum.Real) RETURN Longitude;
Function: calc_lat_long

Input: Latitude, Longitude, Bearing, Range (in Yards)

Output: Latitude, Longitude, Bearing

Description: All Values are Realnum.Real which has 12 digits of precision

Algorithm takes a Bearing and Distance from a known Latitude & Longitude and computes new Latitude and Longitude, Back Bearing. This Algorithm is from Professor John Clynch, Naval Postgraduate school. Original program was written in Fortran and converted to Ada Code. Professor John Clynch stated that this Algorithm has been reviewed and approved by the USGS.

PROCEDURE calc_lat_long (Lat1: IN Latitude; Long1: IN Longitude;
Bearing1: IN Realnum.Real;
Range1: IN Realnum.Real; Lat2: OUT Latitude;
Long2: OUT Longitude;
Bearing2: OUT Realnum.Real);

Function: calc_bearing_dist

Input: Latitude, Longitude, Latitude, Longitude

Output: Bearing, Bearing, Range (in Yards)
--| Description: All Values are Realnum.Real which has 12 digits of precision

--| Algorithm takes a two known Latitude & Longitude's and

--| computes a fwd and back bearing, and Range. This Algorithm was take from

--| Professor John Clynch, Naval Postgraduate school. Original program was

--| written in Fortran and converted to Ada Code. Professor John Clynch stated

--| that this Algorithm has been reviewed and approved by the USGS.

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

PROCEDURE calc_bearing_dist (Lat1: IN Latitude; Long1: IN Longitude;
Lat2: IN Latitude;
Long2: IN Longitude; Bearing1: OUT Realnum.Real;
Bearing2: OUT Realnum.Real;
Range1: OUT Realnum.Real);

--SET and GET LATITUDE PROCEDURE and FUNCTIONS**********************

--**********************************************************************

PROCEDURE Set_Latitude(D : IN Integer; M : IN Integer; S : IN Integer;
128

PROCEDURE Set_Latitude(D : IN Integer; M : IN Integer; S : IN Integer;

128
PROCEDURE Set_Lat_Deg(D : IN Integer; Lat : IN OUT Latitude);

PROCEDURE Set_Lat_Min(M : IN Integer; Lat : IN OUT Latitude);

PROCEDURE Set_Lat_Secs(S : IN Integer; Lat : IN OUT Latitude);
PROCEDURE Set_Lat_Sec(S : IN Integer; Lat : IN OUT Latitude);

PROCEDURE Set_Lat_Sign(Sign : IN Character; Lat : IN OUT Latitude);

FUNCTION Get_Latitude(Lat : Latitude) RETURN Latitude;
FUNCTION Get_Lat_Deg(Lat : Latitude) RETURN Integer;

FUNCTION Get_Lat_Min(Lat : Latitude) RETURN Integer;

FUNCTION Get_Lat_Sec(Lat : Latitude) RETURN Integer;
FUNCTION Get_Lat_Sec(Lat : Latitude) RETURN Integer;

FUNCTION Get_Lat_Sign(Lat : Latitude) RETURN Character;

PROCEDURE Set_Longitude(D : IN Integer; M : IN Integer; S : IN Integer;
PROCEDURE Set_Long_Deg(D : IN Integer; Lon : IN OUT Longitude);

PROCEDURE Set_Long_Min(M : IN Integer; Lon : IN OUT Longitude);

PROCEDURE Set_Long_Sec(S : IN Integer; Lon : IN OUT Longitude);
--| Description: Sets Seconds into Longitude Type
--| Warnings: None

--*******************************************************************************

PROCEDURE Set_Long_Sec(S : IN Integer; Lon : IN OUT Longitude);

--*******************************************************************************

--| Procedure: Set_Long_Sign
--| Input:  Sign: Character; Longitude
--| Output: Longitude
--| Description: Sets Sign into Longitude Type
--| Warnings: None

--*******************************************************************************

PROCEDURE Set_Long_Sign(Sign : IN Character; Lon : IN OUT Longitude);

--*******************************************************************************

--| Function: Get_Longitude
--| Input:  Longitude
--| Output: Longitude
--| Description: Returns a Longitude Type
--| Warnings: None

--*******************************************************************************

FUNCTION Get_Longitude(Lon : Longitude) RETURN Longitude;

--*******************************************************************************
--| Function: Get_Long_Deg
--| Input: Longitude
--| Output: Integer(Degree)
--| Description: Returns a Integer Type
--| Warnings: None

FUNCTION Get_Long_Deg(Lon : Longitude) RETURN Integer;

--| Function: Get_Long_Min
--| Input: Longitude
--| Output: Integer(Minute)
--| Description: Returns a Integer Type
--| Warnings: None

FUNCTION Get_Long_Min(Lon : Longitude) RETURN Integer;

--| Function: Get_Long_Sec
--| Input: Longitude
--| Output: Integer(Second)
--| Description: Returns a Integer Type
--| Warnings: None

FUNCTION Get_Long_Sec(Lon : Longitude) RETURN Integer;
FUNCTION Get_Long_Sec(Lon : Longitude) RETURN Integer;

--*********************************************************
--| Function: Get_Long_Sign
--| Input: Longitude
--| Output: Character(Sign)
--| Description: Returns a Character Type
--| Warnings: None
--*********************************************************

FUNCTION Get_Long_Sign(Lon : Longitude) RETURN Character;

PRIVATE

TYPE Latitude IS RECORD
  latd : Lat_Degree := 0;
  latm : Min_utes := 0;
  lats : Sec_onds := 0;
  AngleSign: Character := 'N';
END RECORD;

TYPE Longitude IS RECORD

136
longd : Long_Degree := 0;
longm : Min_utes := 0;
longs : Sec_onds := 0;
AngleSign : Character := 'W';

END RECORD;

END lat_long;

A-12 UTILITIES.ADS

--****************************************************************
--| FILE: Utilities.ads
--| AUTHOR: Kenneth L. Ehresman and Joey L. Frantzen, NPGS
--| LAST MODIFIED: 11 September 2001
--| OPERATING ENVIRONMENT: Windows 2000(Designed to be O/S Independent)
--| COMPILER: GNAT 3.13p
--| DESCRIPTION: This class is contains all of the data types and
--| functionality for generic general purpose procedures and functions
--| for the navigator program.
--| INPUTS: Based on specific function or procedure.
--| OUTPUTS: Outputs based on what Functions are requested and run.
--| Process: None to note
--| Assumptions: N/A
WITH Realnum;
WITH Ada.Strings; USE Ada.Strings;
WITH Ada.Numerics.Generic_Elementary_Functions;
WITH Ada.Numerics; USE Ada.Numerics;

PACKAGE Utilities IS

TYPE My_Degree IS RANGE 0 .. 359;

-- Defines range of track numbers allowed, 1100..2101 for total of 1000
-- The last slot is never used unless it is an overflow error

LAST_TRACK_SLOT : CONSTANT INTEGER := 2101;
BEGIN_TRACK_SLOT : CONSTANT INTEGER := 1100;

SUBTYPE Track_Number_Range IS Integer RANGE 1100..LAST_TRACK_SLOT;

FUNCTION Validate_Time( T ime : IN Realnum.Real) RETURN integer;
FUNCTION Get_Default_String RETURN String;

PROCEDURE Convert_String_To_Real(Temp_String : String;

    Temp_Number : IN OUT Realnum.Real;

    Okay : IN OUT Boolean   );

FUNCTION Convert_To_Degree_String( Degree_Number : My_Degree )

    RETURN String;

FUNCTION Convert_To_Hours_String( Hours : Integer ) RETURN String;

FUNCTION Convert_Track_Number_To_String( New_Track_Number : Integer )

    RETURN String;

FUNCTION Convert_Integer_To_String( New_Number : Integer )

    RETURN String;

FUNCTION Calculate_Scale ( Hit_Range : Realnum.Real ) RETURN Integer;

END Utilities;
A-13 MOBOARD.ADS

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
--| FILE: moboard.ads
--| AUTHOR: Kenneth L. Ehresman, Naval Post Graduate School
--| LAST MODIFIED: 1 October 2001
--| OPERATING ENVIRONMENT: LINUX & Windows 2000(Designed to be O/S Independent)
--| COMPILER: GNAT 3.13p
--| DESCRIPTION:  This class draws a moboard, and provides functionality for drawing lines,
--| plotting contacts, and refreshing the moboard. It is used by other classes
--| Like CPA, Wind, etc that rely on the digitally-drawn moboard.
--| INPUTS: Function Specific
--| OUTPUTS: None
--| Process: None to note
--| Assumptions: N/A
--| Warnings: None
--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

WITH Ada;                                USE Ada;
WITH Gtk;                                USE Gtk;
WITH Glib;                               USE Glib;
WITH Gdk.Color;
WITH Gdk.Drawable; USE Gdk.Drawable;
WITH Gdk.Gc; USE Gdk.Gc;
WITH Gdk.Pixmap;
WITH Gtk.Drawing_Area;
WITH Ada.Numerics; USE Ada.Numerics;
WITH Degrees;
WITH Lat_Long;
WITH Realnum; USE Realnum;
WITH Sketchpad;
WITH Tracks;
WITH Utilities;

PACKAGE Moboard IS

--| Function: Get_Current_Contact_Num
--| Input: None
--| Output: Returns the active contact Number
FUNCTION Get_Current_Contact_Num RETURN Integer;
--| Preconditions: None
--| Description: See output
--| Side Effects: None

FUNCTION Get_Current_Contact_Bearing RETURN Realnum.Real;
--| Function: Get_Current_Contact_Bearing
--| Input: None
--| Output: Returns the active contact number's current bearing
--| Preconditions: None
--| Description: See output
--| Side Effects: None

FUNCTION Get_Current_Contact_Range RETURN Realnum.Real;
--| Function: Get_Current_Contact_Range
--| Input: None
--| Output: Returns the active contact number's current range in yards
FUNCTION Get_Current_Contact_Range RETURN Realnum.Real;

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

--| Function: Get_Current_Color
--| Input: None
--| Output: Returns the Moboard's current foreground drawing color
--| Preconditions: None
--| Description: See output
--| Side Effects: None

FUNCTION Get_Black_Color RETURN Gdk.Color.Gdk_Color;

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

--| Function: Get_Black_Color
--| Input: None
--| Output: Returns the Moboard's created color black

--| Preconditions: None

--| Description: See output

--| Side Effects: None

FUNCTION Get_Black_Color RETURN Gdk.Color.Gdk_Color;

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

-- Function: Get_Red_Color

--| Input: None

--| Output: Returns the Moboard's created color red

--| Preconditions: None

--| Description: See output

--| Side Effects: None

FUNCTION Get_Red_Color RETURN Gdk.Color.Gdk_Color;

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
|--| Function: Get_Own_Ship_Course
|--| Input: None
|--| Output: Returns own ship course in true degrees as drawn in the moboard center via a speed vector
|--| Preconditions: None
|--| Description: See output
|--| Side Effects: None

FUNCTION Get_Own_Ship_Course RETURN Utilities.My_Degree;

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

|--| Function: Get_Own_Ship_Speed
|--| Input: None
|--| Output: Returns own ship speed in knots as drawn in the moboard center via a speed vector
|--| Preconditions: None
|--| Description: See output
|--| Side Effects: None

FUNCTION Get_Own_Ship_Speed RETURN Realnum.Real;

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
--| Function: Get_Ship_XPos

--| Input: None

--| Output: Returns own ship x-pixel coordinate for the drawn speed vector, called in CPA.Finish_Speed_Triangle

--| Preconditions: None

--| Description: See output

--| Side Effects: None

FUNCTION Get_Ship_XPos RETURN Gint;

--| Function: Get_Ship_YPos

--| Input: None

--| Output: Returns own ship y-pixel coordinate for the drawn speed vector, called in CPA.Finish_Speed_Triangle

--| Preconditions: None

--| Description: See output

--| Side Effects: None

FUNCTION Get_Ship_YPos RETURN Gint;
-- Procedure: Set_Contact_XPos

-- Input: New_XPos - X-pixel position of the contact, used for calculating contacts
course & speed in CPA.Finish_Speed_Triangle

-- Output: None

-- Preconditions: None

-- Description: See input

-- Side Effects: None

PROCEDURE Set_Contact_XPos( New_XPos : Gint );

-- Procedure: Set_Contact_YPos

-- Input: New_YPos - Y-pixel position of the contact, used for calculating contacts
course & speed in CPA.Finish_Speed_Triangle

-- Output: None

-- Preconditions: None

-- Description: See input

-- Side Effects: None

PROCEDURE Set_Contact_YPos( New_YPos : Gint );
--| Procedure: Set_Current_Color
--| Input:  New_Color - the new foreground drawing color
--| Output: None
--| Preconditions: None
--| Description: sets the new foreground drawing color
--| Side Effects:  None

PROCEDURE Set_Current_Color( New_Color : Gdk.Color.Gdk_Color );

--| Procedure: Set_Current_Contact_Bearing
--| Input:  Temp_Bearing - the current bearing of the active contact, used in CPA
--| Output: None
--| Preconditions: None
--| Description: sets the current contact bearing
--| Side Effects:  None

PROCEDURE Set_Current_Contact_Bearing( Temp_Bearing : Realnum.Real );
PROCEDURE Set_Current_Contact_Range( Temp_Range : Realnum.Real );

PROCEDURE Set_Current_Contact_Num

PROCEDURE Set_Current_Contact_Range( Temp_Range : Realnum.Real );

PROCEDURE Set_Current_Contact_Num

PROCEDURE Set_Current_Contact_Range( Temp_Range : Realnum.Real );

PROCEDURE Set_Current_Contact_Num

PROCEDURE Set_Current_Contact_Range( Temp_Range : Realnum.Real );

PROCEDURE Set_Current_Contact_Num
PROCEDURE Set_Current_Contact_Num(Temp_Num : Integer);

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

--| Procedure: Set_Own_Ship_Course

--| Input: New_Course - the new course

--| Output: None

--| Preconditions: None

--| Description: sets the new own ship course

--| Side Effects: None

PROCEDURE Set_Own_Ship_Course(New_Course : Utilities.My_Degree);

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

--| Procedure: Set_Own_Ship_Speed

--| Input: New_Speed - the new speed

--| Output: None

--| Preconditions: None
PROCEDURE Set_Own_Ship_Speed( New_Speed : Realnum.Real );

PROCEDURE Draw_A_Line( X1_Pos : Gint; Y1_Pos : Gint; X2_Pos : Gint; Y2_Pos : Gint );
--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

---| Procedure: Calculate_X_Y_Pixels
---| Input 1  : Input_Bearing - the true bearing of a contact
---| Input 2  : Input_Range - the range of the contact in yards
---| Input 3  : Return_XPos - the calculated x-pixel position

Input 4  : Return_YPos - the calculated y-pixel position
---| Input 5  : The_Scale - the display scale used for ensuring the hit plots to the screen
---| Output:    See description
---| Preconditions:  None

---| Description:    Calculates, displays and returns the X and Y Pixel coordinates of

---                 a contact given

---                 the bearing and range and scale

---| Side Effects:   None

---| Assumptions:    None

PROCEDURE Calculate_X_Y_Pixels( Input_Bearing : Realnum.Real; Input_Range : Realnum.Real;

                        Return_XPos : IN OUT Gint; Return_YPos : IN OUT Gint;
The_Scale : Integer );

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
-- Procedure: Calculate_Slope_Of_Line

-- Input 1 : M1_XPos - The X_Pixel Position of First Hit
-- Input 2 : M2_XPos - The X_Pixel Position of Second Hit
-- Input 3 : M1_YPos - The Y_Pixel Position of First Hit
-- Input 4 : M2_YPos - The Y_Pixel Position of Second Hit
-- Input 5 : X_Fact - The delta of the two X_Positions in Pixels
-- Input 6 : Y_Fact - The delta of the two Y_Positions in Pixels
-- Input 7 : Slope - The ratio of Y_Fact / X_Fact

-- Output: See description

-- Preconditions: None

-- Description: This procedure calculates the slope of a line, as well as the deltas of the
-- X and Y_Positions. Note Parameters 5, 6, and 7 are calculated and returned.

-- Side Effects: None

-- Assumptions: None

PROCEDURE Calculate_Slope_Of_Line( M1_XPos : Gint; M2_XPos : Gint; M1_YPos : Gint; M2_YPos : Gint;

X_Fact: IN OUT Realnum.Real; Y_Fact : IN OUT Realnum.Real;

Slope : IN OUT Realnum.Real );
--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

|-- Function: Find_Distance

|-- Input 1 : M1_X - the x-pixel position of the first point

|-- Input 2 : M2_X - the y-pixel position of the second point

|-- Input 3 : M1_Y - the y-pixel position of the first point

|-- Input 4 : M2_Y - the y-pixel position of the second point

|-- Output: See description

|-- Preconditions: Two valid points must be passed in

|-- Description: Calculates the distance between two points in pixels

|-- Side Effects: None

|-- Assumptions: None

FUNCTION Find_Distance( M1_X : Realnum.Real; M2_X: Realnum.Real; M1_Y : Realnum.Real; M2_Y: Realnum.Real ) RETURN Realnum.Real;

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
--| Input 4 : Print_It - used to determine whether or not to graphically print out the CPA
--| Output:  See description
--| Preconditions:  None
--| Description:  Takes the slope of a line as input and returns the Direction of Relative
Motion
--| Side Effects:  None
--| Assumptions:  None

FUNCTION Find_DRM( Ratio : Realnum.Real; Xfact: Integer; Yfact : Integer; Print_It: Boolean ) RETURN Utilities.My_Degree;

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

--| Procedure: Update_Course_Speed
--| Input 1  : Course_Str - a non-empty string
--| Input 2  : Speed_Str - a non-empty string
--| Output:  See description
--| Preconditions:  None
--| Description:  Converts the inputted srings, checks their validity, and then update both
the Ownship Course and Speed Clist
--| as well as the database
--| Side Effects:  None
--| Assumptions:  None

PROCEDURE Update_Course_Speed( Course_Str : String; Speed_Str : String );
PROCEDURE Draw_Moboard ( New_Area : Sketchpad.Gtk_Sketchpad );

PROCEDURE Init( D_Width: Gint; D_Height: Gint; X_Orig : Gint; Y_Orig : Gint;
    GreenGc : Gdk_GC; BlueGc : Gdk_GC; Dashed_GreenGc : Gdk_GC;
    RedGC : Gdk_GC; New_Area : Sketchpad.Gtk_Sketchpad;
    Red_Color : Gdk.Color.Gdk_Color; Black_Color : Gdk.Color.Gdk_Color  );

PROCEDURE Replenish_Screen;

PROCEDURE Draw_All_Contacts;

-- This procedure draws an arrow at the end of a vector
-- Argument 1: Vector Direction - is the direction the line is going
-- Argument 2: X_Position - is the screen X_Position to center around
-- Argument 3: Y_Position - is the screen Y_Position to center around

PROCEDURE Draw_An_Arrow( Vector_Direction : Realnum.Real; X_Position : Gint; Y_Position : Gint );

PROCEDURE Update.Contact( Contact_Num : String; Contact_ID: String; Contact_Bearing : String;
Contact_Range : String; New_Cxtn : Boolean );

PROCEDURE Plot_A_Point( Plot_Bearing : Realnum.Real; Plot_Range : Realnum.Real; Scale : Integer );

-- This procedure saves the own ship Longitude
-- Argument 1: valid longitude degrees
-- Argument 2: valid longitude minutes
-- Argument 3: valid longitude seconds
-- Argument 4: valid longitude hemisphere

PROCEDURE Update_Longitude( Long_Degrees : Lat_Long.Long_Degree; Long_Minutes : Lat_Long.Min_utes;
PROCEDURE Update_The_Latitude( Lat_Degrees : Lat_Long.Long_Degree;
Lat_Minutes : Lat_Long.Minutes;
Lat_Seconds : Lat_Long.Seconds;  Lat_Hemisphere :
String );

-- This procedure saves the own ship Latitude

-- Argument 1: valid latitude degrees
-- Argument 2: valid latitude minutes
-- Argument 3: valid latitude seconds
-- Argument 4: valide latitude hemisphere

PRIVATE

Current_Contact_Num : Integer;

Current_Contact_Bearing,
Current_CONTACT_Range,
Moboard_Radius : Realnum.Real;

Black,
Current_Color,
Red : Gdk.Color.Gdk_Color;

Ratio_Array : ARRAY(0..2, Utilities.My_Degree) OF Realnum.Real;

NUMBER_O_CIRCLES : CONSTANT Integer := 11;

One_Degree : Realnum.Real := Pi / 180.0;
Own_Ship_Course : Utilities.My_Degree := 0;
Own_Ship_Speed : Realnum.Real := 0.0;
Set_Up_Distance : Realnum.Real := 100_000.0;

Current_Area : Sketchpad.Gtk_Sketchpad;

Contact_XPos,
Contact_YPos,
Ship_XPos,
Ship_YPos,
Draw_Width,
PROCEDURE Initialize_Ratio_Array;

-- Procedure: Initialize_Ratio_Array
-- Input: None
-- Output: None
-- Preconditions: None
-- Description: This procedure calculates the ratio of Y over X for determining a unique slope and value for every degree from 0 to 359 degrees. The values are stored in Ratio_Array for use in determining which degree the slope of a line represents.
-- Side Effects: Ratio_Array values are set and calculated using a range of 100000 yards in order to make them distinctly unique.
--- Procedure: Plot_Course_Vector

--- Input 1: Ship_Course - A Realnum. Real number representing Own Ships Course in degrees

--- Input 2: Ship_Speed - A Realnum. Real number representing Own Ships Speed in knots

--- Output: See description

--- Preconditions: None

--- Description: Draws the course and speed vector for the ship

--- Side Effects: None

--- Assumptions: None

PROCEDURE Plot_Course_Vector( Ship_Course : Realnum.Real; Ship_Speed : Realnum.Real );
WITH Ada; USE Ada;
WITH Gtk; USE Gtk;
WITH Glib; USE Glib;

WITH Gdk.Color;
WITH Gdk.Drawable; USE Gdk.Drawable;
WITH Gdk.Gc; USE Gdk.Gc;
WITH Gdk.Pixmap;
WITH Gtk.Drawing_Area;

WITH Ada.Numerics; USE Ada.Numerics;


WITH Degrees;

WITH Realnum; USE Realnum;

WITH Sketchpad;

WITH Times;

WITH Tracks;

WITH Utilities;

PACKAGE cpa IS

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

--| Function: Get_Alert_Distance

--| Input: None

--| Output: Returns the user-defined CPA alert distance

--| Preconditions: None

--| Description: See output

--| Side Effects: None
FUNCTION Get_Alert_Distance RETURN Realnum.Real;

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

|-- Procedure: Set_Alert_Distance

|-- Input: A real number, representing the new user-defined CPA alert distance

|-- Output: None

|-- Preconditions: None

|-- Description: Sets the CPA_Alert_Distance to a new distance selected from a ComboBox by the user

|-- Side Effects: None

PROCEDURE Set_Alert_Distance( New_Distance : Realnum.Real );

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

|-- Procedure: Calculate_CPA

|-- Input 1 : Track_Num - The track number of the contact whose CPA is to be calculated

|-- Output: The selected contacts graphically generated CPA, as well as an update to the screen of the CPA details in

|-- the Contact CPA Clist

|-- Preconditions: The track number must have two Radar hits.

|-- Description: This procedure uses the last two Radar hits of a contact, and graphically draws the CPA. While drawing

|-- CPA the SRM and MRM are calculated, the speed triangle is generated, and the contact's current course,
speed, and all CPA details are calculated and displayed. Upon completion
the boolean in the Sketchpad

class which allows manual fairing of all bearings is set to true, thereby
enabling the user's ability to

recalculated the Course, Speed and CPA of the contact using faired bearings.
The contacts last two radar hits are plotted and the Measure of Relative
Motion (MRM)
is calculated in yards.

Next, the Direction of Relative Motion (DRM) is calculated and the slope of
the inter-
connected line between the two hits is determined.

A line projection using the DRM is drawn to the edge of the outer-moboard
ring, and the CPA

is calculated by the closest part of this line to the Moboard's origin (center).
The CPA Bearing, Time, and Range are then displayed, and the contact's
course and speed

are determined by calling the procedure "Finish_Speed_Triangle"

Side Effects: The current contact's CPA, course and speed are saved into the Track
Class.

Sketchpad's Boolean allowing the fairing of bearings is set to True.

Assumptions: None

PROCEDURE Calculate_CPA( Track_Num : Utilities.Track_Number_Range );

******************************************************************************
**Procedure: Initialize**

---

**Input 1**: Current_Area - The double-buffered drawing area maintained in the Sketchpad class

**Input 2**: Current_X_Origin - The x-pixel position of the center of the drawing area

**Input 3**: Current_Y_Origin - The y-pixel position of the center of the drawing area

---

**Output**: None

---

**Preconditions**: A current drawing area in the Sketchpad class has been instantiated.

---

**Description**: This procedure is passed in the double-buffered drawing area maintained in the Sketchpad class, as well as the X and Y pixel coordinates of the center of the drawing area.

---

**Side Effects**: None

---

**Assumptions**: None

---

```lisp
PROCEDURE Initialize( Current_Area : Sketchpad.Gtk_Sketchpad; Current_X_Origin : Gint; Current_Y_Origin : Gint );
```

---

**Procedure: Manually_Fair_Bearings**

---

**Input 1**: X_Pos - The x-pixel position chosen by left clicking the mouse
PROCEDURE Manually_Fair_Bearings( X_Pos : Gint; Y_Pos : Gint );

--| Procedure: Initialize_Faired_Bearing_Variables

--| Input 1  : Track_Num - The track number of the contact to Fair Bearings on
--| Input 2  : Track_Bearing1 - The contact's first bearing to be faired in degrees
--| Input 3  : Track_Range1 - The contact's first range in yards
--| Input 4  : X1_Pos - The x-pos pixel position of the first bearing
--| Input 5  : Y1_Pos - The y-pos pixel position of the first bearing
PROCEDURE Initialize_Faired_Bearing_Variables( Track_Num : Integer;
Track_Bearing1 : Utilities.My_Degree;
Track_Range1 : Realnum.Real; X1_Pos : Gint; Y1_Pos : Gint;
Scale : Integer; Track_Bearing2 : Utilities.My_Degree;
Track_Range2 : Realnum.Real; X2_Pos : Gint; Y2_Pos : Gint; Time1 : Times.Time_Type; Time2 : Times.Time_Type );

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
PRIVATE

SPEED_SCALE : CONSTANT Realnum.Real := 5.0;
NUMBER_O_CIRCLES : CONSTANT Integer := 11;

One_Degree : Realnum.Real := Pi / 180.0;

CPA_Drawing_Area : Sketchpad.Gtk_Sketchpad;

X-Origin,
Y-Origin : Gint;

CPA_Alert_Distance : Realnum.Real := 10000.0;

-- The following variables are for fairing the bearings.

Faired_First_XPos,
Faired_First_YPos,
Faired_Second_XPos,
Faired_Second_YPos : Gint;

Faired_First_Bearing,
Faired_Second_Bearing : Utilities.My_Degree;
Faired_First_Range,
Faired_Second_Range : Realnum.Real;

Faired_Scale,
Faired_Track_Num : Integer;

Faired_Time1,
Faired_Time2 : Times.Time_Type;

--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

|-- Procedure: Calculate_SRM
|--| Input 1 : MRM - Measure of Relative Motion in yards
|--| Input 2 : M1_Time - Time of First Hit
|--| Input 3 : M2_Time - Time of Second Hit
|--| Input 4 : First_Hit_Time - Time of First Hit converted to just Hours and minutes
|--| Input 5 : SRM - the Speed of Relative Motion in knots
|--| Input 6 : Valid - a boolean that indicates if the calculated SRM is valid and therefore usable

|-- Output: The SRM is calculated and returned, as well as the Boolean value Valid, which indicates if the SRM is usable

|-- Preconditions: Two RADAR hists must be available as well as the Measure of Relative motion. Each RADAR hit must have an
associated time, which can not be identical. Additionally, the calculated SRM must be > 0.0 knots to be valid.

Description: This procedure takes the measure of relative motion, measured in yards, and returns the Speed of Relative motion in knots.

PROCEDURE Calculate_SRM( MRM : Realnum.Real; M1_Time : Times.Time_Type; M2_Time : Times.Time_Type;
First_Hit_Time : IN OUT Realnum.Real; SRM : IN OUT Realnum.Real; Valid : IN OUT Boolean );

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

Procedure: Finish_Speed_Triangle

Input 1 : SRM - the speed of relative motion in knots
Input 2 : DRM - the direction of relative motion in Degrees
Input 3 : Contact_Number - the contact number of input contact

Output: Graphically displayed as per Description. ( Contact's Course and Speed vector as well as relative DRM )

Preconditions: This procedure is called by Calculate_CPA; therefore, all of Calculate_CPA's post-conditions must be met

Description: This procedure takes as input the speed of relative motion and direction of relative motion, which are used for drawing the speed triangle, just like a regular moboard does. During procedure instantiation, the
scale for the contact number is retrieved from the Tracks class, as well as the X and Y pixel coordinates of own ship's speed vector (via Moboard class).

The contact's course and speed are graphically drawn, then inferred from the drawing, and saved via the Tracks class.

Side Effects: The Track_Num's calculated course and speed are updated.

PROCEDURE Finish_Speed_Triangle( SRM : Realnum.Real; DRM : Utilities.My_Degree; Track_Num : Utilities.Track_Number_Range );

END cpa;

A-15 MAINSCREEN-PKG.ADS

WITH Gtk.Arguments;

WITH Gtk.Widget; USE Gtk.Widget;

WITH Gtk.Color_Selection_Dialog; USE Gtk.Color_Selection_Dialog;
WITH Gtk.Text; USE Gtk.Text;

WITH Gtk.Window; USE Gtk.Window;

WITH Gtk.Button;

WITH Gtk.Menu_Item;

PACKAGE Main_Screen_Pkg.Callbacks IS

Text1  : Gtk_Text;

Show_Course: Boolean := True;

Color.Done : Boolean := False;

-- The first three procedures must be instantiated, even though they will never be connected!

PROCEDURE On_Cpa1_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);
PROCEDURE On_Preferences1_Activate(Object: ACCESSGtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Contact1_Activate(Object: ACCESSGtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Show_Closest_Activate(Object: ACCESSGtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE Put_Lat_Long_Into_Ownship_Lat_Long_Clist;

-- Manually fair the bearing to recalculate the CPA

PROCEDURE On_Manual_Fair_Activate(Object: ACCESSGtk.Menu_Item.Gtk_Menu_Item_Record'Class);

--The following will eventually be implemented
PROCEDURE On_Show_Cpas_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Show_Lines_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Update_Contact_Info_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Course1_Text_Changed( Object : ACCESS Gtk_Text_Record'Class );
the following procedures have been implemented

PROCEDURE On_Drop_Contact_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

-- The next three are for creating a new contact, and responding to the submit and Cancel buttons

PROCEDURE On_Create_New_Contact_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_New_Contact_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE On_New_Contact_ButonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

-- the next three are for inputting a new hit, and responding to its submit and Cancel buttons
PROCEDURE On_Input_New_Contact_Hit_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Input_Hit_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE On_Input_Hit_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

-- The next three are for inputting own ships course and speed

PROCEDURE On_Input_Course_Speed_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Course_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE On_Course_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );
PROCEDURE On_Input_Lat_Long_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Lat_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class);

PROCEDURE On_Lat_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class);

PROCEDURE On_Color_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class);

PROCEDURE On_Color_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class);
PROCEDURE Put_Lat_Long_Into_Track_Lat_Long_Clist( Track_Number : Integer );

PROCEDURE Change_The_Contacts_Colors;

-- Process the wind options

PROCEDURE   On_Wind_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class );

PROCEDURE   On_True_Wind_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class );

PROCEDURE   On_Wind_Course_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class );

PROCEDURE   On_Wind_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE   On_Wind_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

--Process all contact purge times

PROCEDURE   On_Track_Purge_Time_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class );
PROCEDURE On_Purge_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE On_Purge_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

-- ************************************************************
-- This procedure checks the number of currently held contacts in the database, and deletes
-- any extraneous tracks that are being displayed in the contact_clist. In other words,
-- it keeps the screen current with tracks that are dropped by the db due to age

PROCEDURE Purge_The_Contact_Clist;

-- Process the setting of the Minimum_CPA_Alert_Range

PROCEDURE On_CPA_Alert_Range_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class );

PROCEDURE On_Alert_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE On_Alert_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

END Main_Screen_Pkg.Call backs;
PACKAGE Main_Screen_Pkg.Callbacks IS

  Text1  : Gtk_Text;

  Show_Course: Boolean := True;

  Color_Done : Boolean := False;
-- The first three procedures must be instantiated, even though they will never be connected!

PROCEDURE On_Cpa1_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Preferences1_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Contact1_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Show_Closest_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE Put_Lat_Long_Into_Ownship_Lat_Long_Clist;
-- Manually fair the bearing to recalculate the CPA

PROCEDURE On_Manual_Fair_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

--The following will eventually be implemented

PROCEDURE On_Show_Cpas_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Show_Lines_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Update_Contact_Info_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Course1_Text_Changed( Object : ACCESS Gtk.Text_Record'Class );

-- the following procedures have been implemented
PROCEDURE On_Drop_Contact_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

-- The next three are for creating a new contact, and responding to the submit and Cancel buttons

PROCEDURE On_Create_New_Contact_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_New_Contact_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE On_New_Contact_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

-- the next three are for inputting a new hit, and responding to its submit and Cancel buttons

PROCEDURE On_Input_New_Contact_Hit_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);
PROCEDURE On_Input_Hit_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE On_Input_Hit_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

-- The next three are for inputting own ships course and speed

PROCEDURE On_Input_Course_Speed_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class );

PROCEDURE On_Course_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE On_Course_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE On_Input_Lat_Long_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Lat_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE On_Lat_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE On_Color_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE On_Color_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE Put_Lat_Long_Into_Track_Lat_Long_Clist( Track_Number : Integer );

PROCEDURE Change_The_Contacts_Colors;

-- Process the wind options
PROCEDURE On_Wind_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_True_Wind_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Wind_Course_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Wind_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class);

PROCEDURE On_Wind_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class);

PROCEDURE On_Track_Purge_Time_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Purge_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class);

PROCEDURE On_Purge_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class);

--Process all contact purge times

PROCEDURE On_Track_Purge_Time_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Purge_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class);

PROCEDURE On_Purge_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class);

-- **********
PROCEDURE Purge_The_Contact_Clist;

PROCEDURE On_CPA_Alert_Range_Activate( Object : ACCESS Gtk.Menu_Item.Gtk_Menu_Item_Record'Class);

PROCEDURE On_Alert_Button_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

PROCEDURE On_Alert_ButtonCancel_Clicked( Object : ACCESS Gtk.Button.Gtk_Button_Record'Class );

END Main_Screen_Pkg.Callbacks;

A-17 SKETCHPAD.ADS

WITH Glib; USE Glib;

WITH Gdk.Color;

WITH Gdk.Drawables;
WITH Gdk.Gc;

WITH Gdk.Pixmap;

WITH Gdk.Window;

WITH Gdk.Drawing_Area;

PACKAGE Sketchpad IS

  Draw_Width,
  Draw_Height,

  X_Origin,

  Y_Origin : Gint;

  type Gtk_Sketchpad_Record is new
    Gdk.Drawing_Area.Gtk_Drawing_Area_Record with private;

  type Gtk_Sketchpad is access all Gtk_Sketchpad_Record'Class;
PROCEDURE Gtk_New( Drawing_Widget : Out Gtk_Sketchpad; Video_Width : Gint; Video_Height : Gint );

PROCEDURE Initialize( Drawing_Widget : ACCESS Gtk_Sketchpad_Record'Class; Video_Width : Gint; Video_Height : Gint );

PROCEDURE Set_Allow_CPA( Enable_CPA : Boolean );
PROCEDURE Allow_Drawing_Fairline( Allow_Drawing : Boolean );
PROCEDURE Set_Anchor_Pixels( X1_Pos : Gint; Y1_Pos : Gint );
FUNCTION Get_Allow_CPA RETURN Boolean;
FUNCTION Get_Moboard_Pixmap( Drawing_Widget : ACCESS Gtk_Sketchpad_Record'Class ) RETURN Gdk.Drawable.Gdk_Drawable;

PRIVATE

TYPE Gtk_Sketchpad_Record IS NEW
Gtk.Drawing_Area.Gtk_Drawing_Area_Record

WITH RECORD

-- The pixmap used for double buffering


END RECORD;

Calculate_CPA       : Boolean := False;
Permission            : Boolean := False;

Faired_XPos_Anchor,
Faired_YPos_Anchor    : Gint;

END Sketchpad;

A-18 WIND.ADS

-- Filename: wind.ads
-- Author: Ken Ehresman
-- This file is for calculating true wind, and desired wind as well as plotting their
-- solutions onto a moboard.
WITH Ada; USE Ada;
WITH Gtk; USE Gtk;
WITH Glib; USE Glib;
WITH Gdk.Color;
WITH Gdk.Drawable; USE Gdk.Drawable;
WITH Gdk.Gc; USE Gdk.Gc;
WITH Gdk.Pixmap;
WITH Gtk.Drawing_Area;
WITH Ada.Numerics; USE Ada.Numerics;
WITH Degrees;
WITH Realnum; USE Realnum;
WITH Sketchpad;
WITH Tracks;
WITH Utilities;

PACKAGE Wind IS

    PROCEDURE Calculate_True_Wind;
-- This procedure is called in order to initialize class variables. It is called from within Moboard's Init procedure

PROCEDURE Initialize( Current_Area : Sketchpad.Gtk_Sketchpad; Current_X_Origin : Gint; Current_Y_Origin : Gint );

-- This procedure is called in order calculate the True Wind from measured wind
-- Argument 1: Measured_Wind_Direction - Direction from which wind was measured from
-- Argument 2: Measure_Wind_Speed     - Speed or measured wind
-- Argument 3: Measured_Wind_Type     - Relative or Apparant wind

PROCEDURE  Calculate_True_Wind( Measured_Wind_Direction : Integer; Measured_Wind_Speed : Integer; Relative_Wind_Type : Boolean );

PRIVATE

Wind_Drawing_Area : Sketchpad.Gtk_Sketchpad;
SPEED_SCALE : CONSTANT Realnum.Real := 5.0;
NUMBER_O_CIRCLES : CONSTANT Integer := 11;
One_Degree : Realnum.Real := Pi / 180.0;

X_Origin,
Y_Origin : Gint;

END Wind;
LIST OF REFERENCES

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center  
   Ft. Belvoir, VA  22060-6218

2. Dudley Knox Library  
   Naval Postgraduate School  
   Monterey, CA  93943-5101

3. Dr. Luqi  
   Naval Postgraduate School  
   luqi@cs.nps.navy.mil

4. Professor Richard Riehle  
   Naval Postgraduate School  
   rdriehle@nps.navy.mil

5. Professor Mantak Shing  
   Naval Postgraduate School  
   shing@nps.navy.mil