AUTOMATED VESSEL LOGS - VOLUME 2: TECHNICAL ASPECTS OF THE LOG PROTOTYPE

WA-RD 296.2

Final Report
June 1993

Washington State Department of Transportation

Washington State Transportation Commission
Transit, Research, and Intermodal Planning (TRIP) Division

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO.	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.
WA-RD 296.2		

4. TITLE AND SUBTITLE	5. REPORT DATE
AUTOMATED VESSEL LOGS — VOLUME 2: TECHNICAL ASPECTS OF THE LOG PROTOTYPE	June 1993
	6. PERFORMING ORGANIZATION CODE

7. AUTHOR(S)	8. PERFORMING ORGANIZATION REPORT NO.
Erik Halvard Beck and Mark Hallenbeck	

9. PERFORMING ORGANIZATION NAME AND ADDRESS	10. WORK UNIT NO.
Washington State Transportation Center (TRAC) University of Washington, JD-10 University District Building; 1107 NE 45th Street, Suite 535 Seattle, Washington 98105-4631	11. CONTRACT OR GRANT NO. T9233, Task 31

12. SPONSORING AGENCY NAME AND ADDRESS	13. TYPE OF REPORT AND PERIOD COVERED
Washington State Department of Transportation Transportation Building, MS 7370 Olympia, Washington 98504-7370	Final report
	14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

16. ABSTRACT

This project developed a prototype computer-aided vessel log system for the Washington State Ferry System (WSF). The researchers generated three reports that describe the results of their research. This report, which is the second volume of three, contains a two-part guide that describes that prototype software program in detail. Part One of this second volume was written for technical administrators who must understand the program's production to enable them to direct refinement of the prototype. Part Two of this volume was written for the programmers who will develop the code refinements. This volume also contains the source code listings for all of this project's programs.

The other two volumes are a summary report for the project and a user's guide. The first volume (the summary report) contains a description of the project and summarizes the design and testing results of the prototype automated vessel log. The third volume contains a user's guide for the prototype software. A diskette containing all of the program's source code and the executable programs has been sent to the WSF Service Planning Manager at Colman Dock

17. KEY WORDS	18. DISTRIBUTION STATEMENT
Ferry operations, vessel logbooks	No restrictions. This document is available to the public through the National Technical Information Service, Springfield, VA 22616

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
None	None	190	

Final Report
Research Project T9233, Task 31
Automated Vessel Logbooks

AUTOMATED VESSEL LOGS — VOLUME 2:

TECHNICAL ASPECTS OF THE LOG PROTOTYPE

by

Erik Halvard Beck
Research Assistant

Mark Hallenbeck
Deputy Director

Washington State Transportation Center (TRAC)
University of Washington, JD-10
University District Building
1107 NE 45th Street, Suite 535
Seattle, Washington 98105-4631

Washington State Department of Transportation
Technical Monitor
David Remagen
WSF Service Planning Manager
Washington State Ferry system

Prepared for

Washington State Transportation Commission
Department of Transportation

June 1993

DISCLAIMER

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the Washington State Transportation Commission or the Department of Transportation. This report does not constitute a standard, specification, or regulation.

TABLE OF CONTENTS

INTRODUCTION

This document is the second volume of a report that describes the development of a prototype computer aided vessel log system for the Washington State Ferry System (WSF). This volume provides technical information about the computer aided vessel log (computerized logbook). The report is divided into two parts. Part One gives information about the operating and other characteristics of the computer system appropriate for a technical supervisor. Part Two gives more detailed information about how the computer system software was programmed and includes the complete programming code for the system. Part Two assumes that the reader is a programmer and is familiar with programming terms and concepts.

PART ONE

OPERATING CHARACTERISTICS

INTRODUCTION

This is Part One of the second volume of the report that describes the development of the prototype computer aided vessel logbook system for the Washington State Ferries. This half of this volume provides information on the operating and other characteristics of the computer system appropriate for a technical supervisor. It is divided into two basic sections. The first sections describes the hardware requirements of a computerized vessel log as it must operate on the WSF vessels. The second section describes the operating requirements of the system, and how those operating requirements were met in the software design. Both sections of the report include discussions of how the current prototype can be improved given additional resources.

HARDWARE

Besides the general design requirements that were described in Volume One (compactness, ruggedness, ease of maintenance, reliability, ease of manipulation, and quiet operation) the computer system hardware also needed to meet certain technical requirements in order to function correctly as a prototype and to accommodate potential future modifications.

The main technical requirement is that the computer system must be capable of running MS-Windows 3.1 or higher. A system capable of running Windows provides a great deal of design and operating flexibility over one that is not capable of properly running Windows. For example, running the FerryLog software under Windows[1] enables the bridge crew to be able to switch at any time to another program, such as a word

---

[1]Please note that if the FerryLog program is run under Windows, it must be run in full screen mode for proper operation. Otherwise, the mouse will not work properly.

processor, without interrupting the FerryLog program or the data that it is collecting. Once the word processing task is complete, the computer system operator can then switch back to the FerryLog program and resume its use without loss of data.

In order to use Windows 3.1, the computer system must have a substantial amount of computer memory (RAM) and hard disk space. Given the current characteristics of Windows and of the FerryLog program, the computer should contain at least 60 Mb of hard disk space and least 4 Mb of RAM, and should use an Intel 80386 SX or better processor. If an 80386 processor is used, it should run at a clock speed of at least 20 Mhz. While 4 Mb of RAM is sufficient to run the FerryLog software with Windows, the computer should be expandable to at least 10 Mb of memory, in order to provide flexibility for future growth in the program or in the other uses WSF crews may have for a computer on the bridge.

For Windows compatibility as well as visibility in high glare situations, the computer should have a high quality VGA compatible monitor with resolution of 640x480 pixels, with a minimum of 16 color levels (or shades of gray). If a monochrome monitor must be used, as a minimum, it should be capable of 64 shades of gray in 320X200 mode. In addition, for good visibility during high glare conditions, the screen should be physically large.

If a laptop or notebook computer is used as the primary computer for the automated logbook, it is recommended that the computer have built in facilities for an external color VGA monitor. As mentioned in Volume One of this report, an external color monitor could improve the visibility of the computer system at a modest cost.

In order to meet current and future requirements, the computer system must also be capable of simultaneously supporting:

- a parallel printer,
- a serial or bus mouse, and
- an internal or external modem.

These capabilities provide the flexibility necessary to ensure compatibility with a wireless computer network proposed to operate between Colman Dock and the WSF vessels.

SOFTWARE

Overview

The computer software was written using the *Borland C++ 3.1 with Application Frameworks* compiler and software tools package. This package was used because of the accessibility of all source code and because it included the *Turbo Vision* class library. This library was used to create the graphical user interface in the FerryLog program, as well as to handle data. The Turbo Vision library was also used in the Combine program.

It is strongly suggested that both the FerryLog and the Combine program be run from within the batch files written for each program. These batch files are included on the distribution diskettes given to WSF. The text for the batch files is also included in the program code appendices attached to this report.

Use of the batch programs to start the Ferrylog and Combine programs allows certain system parameters, such as time and date, to be modified as required. It also allows certain idiosyncrasies of the two programs to be hidden from the end user, promoting proper operation of the programs.

Combine Program

Program Description

The primary function of the Combine program is to take Ferrylog raw data files from each end of a specific WSF vessel and combine them into a single, ASCII, tab delimited, text file in an easy-to-read format. This program is character based, and requires that the files to be read and written be specified on the command line. The program checks that the two files are from the same ferry vessel, are not from the same wheel house, were both created within a reasonable amount of time of each other (that is, the files aren't from different days), and that the route reported in each file is the same. These checks are made by examining the input file names and by examining the contents of the input files. The

input files, as created by the FerryLog software, each have a unique name that indicates when and where they were created. (For more information, see the FerryLog section below).

Potential Improvements To The Combine Program

The primary improvement that could be made to the Combine program is to redesign it so that it will more readily interface with future WSF databases. The prototype Combine program was designed to produce a simple text file that could be read with Microsoft's Excel or some other spreadsheet program. A final version of the Combine program should write data in a format that can be easily incorporated into whatever database the WSF operations staff decides to use.

Another improvement that can be made is to change the character user interface of the Combine program into a graphical user interface. This would give the program increased functionality and ease of use. Finally, the program should be changed to allow the use of the Combine program with logbooks from the passenger only vessels. Currently Combine requires two input files be given on the command line (one file is needed from each wheel house.) This operation characteristic is not appropriate for passenger only vessels which have only one wheel house.

FerryLog Program

Program Description

The primary function of the FerryLog program is to collect information of interest about activities aboard WSF vessels from bridge crews, then organize this information and record it electronically on standard computer disks. This program is designed to be run interactively on computers located on the bridge in each wheel house of WSF vessels. It is designed to replace the paper logbooks currently in use on WSF vessels.

The Ferrylog program is currently designed to capture information on arrivals, departures, checkpoints, incidents, drills, and weather. For all information recorded, the FerryLog program attaches two time stamps. One stamp is the time that the information

was added by the program user. This time is taken from the computer's system time. The other stamp is the time that a user wished to have attached to that information. Usually, these times will be identical. However, there are times that the user may want to attach a time to an action that may be different from the computer's system time. An example of this situation might be an accident or rescue at sea, where the crew is too busy reacting to the incident to immediately enter the incident into the logbook. As an example, this provision allows the Master or Mate to make an entry in the logbook that essentially states, "at 11:00, I entered the fact that I responded to a ship in distress call at 10:45."

In addition to recording this information on electronic disks, the FerryLog program also allows the information being collected to be printed on paper. The log information can be printed in two formats, Print-As-You-Go and Summary. As the names suggest, Print-As-You-Go allows a paper record to be produced during the course of recording information. The Summary printing function allows a paper copy to be produced after all information has been input to the system at the end of a crew's shift. The Summary format contains space for the signatures of the Master and Chief Mate. The Print-As-You-Go paper copy does not have a specific location for these signatures. Both printing formats can be turned on or off by the system operators.

The FerryLog program is a MS-DOS program that uses buttons, pop-up windows, and pull down menus to make it easy to use. The graphical users interface was designed around the way the paper logbook is currently used by WSF bridge crews in order to make it easier for these individuals to learn the system, and to ease their transition from the existing system to the computer based system.

The FerryLog program records information electronically by creating files in two places: on the hard drive, and the main floppy disk drive, (drive 'A'). It creates these files in the following sequence. First, the program creates a temporary file on the hard disk. During an active session (that is when the Ferrylog software is running), all information is recorded to this file. At the end of a session, the contents of this file are copied to a new

file on the hard disk, then to a file on the floppy disk in drive 'A'. If the program can't write to drive 'A', it indicates the problem and repeatedly requests that the problem be relieved (for example by placing a new disk Drive A) until it can write successfully to that drive. Files created by the program have a unique name that follows the following format:

MMDDYYHE.VVC

Where: M is month

D is day

Y is year

H is hour or increment number (A-X, 0-9, these are further explained below)

E is the vessel end number from which the data was collected (0,1, or 2)

VV is the two letter vessel abbreviation (uniquely identifies each vessel), and

C is the file code type.(T or U for temporary file, F or G for final file)

The FerryLog program will first try to write a file based on the date and time using the temporary code, T. The hour field is determined by the hour of the day on a 24 hour clock (00:23 is A, 03:00 is D, 13:45 is N, etc.). If a file with the intended name exists, the program will instead try to write a file with the same name, with the exception that the T is changed to a U, and the hour number becomes the numeral 0. If this file already exists, then the program will try to write a file using the numeral 1 for the hour code, and so on. This file naming system was developed in case the computer has problems, and the software program must be restarted several times during a shift.

At the end of the shift when all information to be recorded is contained in the temporary file, the program will then attempt to write another copy to the hard drive and the floppy disk using the same naming convention, except using F or G as the file code type. For example, a temporary file started at noon on Christmas day in the number one wheel

house on the Walla Walla would be named 122592M1.WWT, while the final copy of this file would be named 122592M1.WWF. If these files already existed, then the first attempt names would be 12259201.WWU and 12259201.WWG, respectively. The second attempt names would be 12259211.WWU and 12259211.WWG, and so on.

In order to do properly derive the code for the wheel house vessel and end, the FerryLog software needs to find the file "FERRYID.DAT". This file contains three characters that identify each vessel in the WSF fleet and wheel house where the system is supposed to reside. The first two characters are the two letter vessel abbreviation (such as WW for Walla Walla). The third character is the wheel house number, 1, 2 or 0, where 0 represents a vessel with only one wheel house. (The file FEERYID.DAT is provided on the diskette which contains the operational Ferrylog program.) If this file cannot be found, the program defaults to a setting of UK9 (Unknown). As the computer system is currently setup, the FERRYID.DAT has file attributes set to "read only" and "hidden file" so that it cannot be inadvertently erased or modified.

Proposed Improvements To The Prototype

The most important improvement that should be made to the FerryLog program is to eliminate the termination bug on the Compaq LTE 20's. As noted in volume one of this report, this bug occurs only on the Compaq portable computers purchased for this project and does not occur on other computers that were used to test the FerryLog software. This bug occurs at the end of the program's operation, after all information has been stored and transferred to the floppy disk. After all information has been transferred, under normal operation the program will terminate, the screen will be cleared, and the system prompt (C:>) will return. On the Compaq portable computers this fails to happen. Instead, the screen fails to be cleared, and the system prompt does not return. Essentially, the computer has halted operation. In order to continue to use the computer, a soft boot (pressing the Ctrl, Alt , and Delete keys simultaneously) must be performed.

For the moment, this termination problem can be alleviated by running FerryLog under MS Windows 3.1, and utilizing the "force exit" facility built in to Windows' DOS task swapper. This facility forces the FerryLog program to quit running in and returns the computer to the DOS prompt as intended. This "work around" will allow the system to continue working normally until a more elegant solution can be developed.

The next most important improvement that should be undertaken is to expand the functionality of the system from the Seattle--Bainbridge Island route to all WSF routes and vessels. As mentioned in Volume one, the FerryLog program needs some minor revisions to be usable on all WSF routes and on all WSF vessels . The prototype computer system is currently designed for the way the log book is used on vessels with two wheel houses running a simple two point route. To expand its usefulness to the passenger only vessels with one wheel house or to more complex routes such as those in the San Juan Islands, the operating characteristics of the users interface needs to be modified to allow for the variety of operational situations these routes present. For example, arrival, departure, and checkpoint names for all routes other than the Seattle--Bainbridge Island must be added to the program. Currently, when a route other than the Seattle--Bainbridge Island route is selected, all point names are labeled as "generic" or "blank". these should be replaced by the names appropriate for each route. For multi-point routes, it is also important that the Master and Mate be allowed to change the order in which these points are reached, because vessels do not always follow a consistent pattern of terminal stops. (That is, on some trips the ferry may stop at Lopez Island, and on others it may not. Thus the markers entering the Lopez harbor should be present in some logbook entries, and not in others,)

The third addition to the prototype is a facility for entering offenses by seamen into the logbook. These logbook entries should be made in accordance with *Title 46, United States Code, section 11502*. When adding this new capability to the program, care should be taken to satisfy not only the USCG regulations, but also the corporate requirements of WSF (i.e., any special data collection requirements that need to be met to meet WSF

specific union or Washington State regulatory laws.) If properly written, this function could replace the need for the master and chief mate to fill out a separate Econogram detailing the offense committed.

Other additions that should be made to the interface include a facility for unusual runs such as a fueling run to Harbor Island or a vessel transfer run from Eagle Harbor to one of the routes. Also when a fueling run is made, USCG regulations require that an entry must be made in the logbook recording the amount of fuel on hand and the amount received. This capability requires minor revisions to the FerryLog interface as well as minor changes to the basic data storage routines.

Finally, at some point, a facility for archiving and erasing old log information from each computer hard drive should also be implemented. This would help safeguard against the loss of important log information, and would prevent excessive file buildup on the hard disk. (Note that the file copy on the hard drive serves as a back-up to the floppy disk copy, until the files on the hard disk drive are removed.

PART TWO

PROGRAMMERS GUIDE

INTRODUCTION

This is Part Two of the second volume of the report that describes the development of the prototype computer aided vessel logbook system for the Washington State Ferries. This half of this volume should be used as a reference by WSF staff responsible for maintaining or improving the automated vessel logbook software, and serves as the documentation for the software code. This section presumes a basic knowledge of the C and C++ programming languages.

COMPILER

The compiler used to create all automated logbook software is *Borland C++ 3.1 with Application Frameworks*. This compiler package was chosen because of the *Turbo Vision* class library that was included in the package, as well as the availability of the source code for all library functions and class libraries. Extensive use was made of the Turbo Vision classes in the prototype software programs.

This compiler is actually several compilers in one package. There are ANSI C and C++ (AT&T 2.1 Full Implementation) compilers for MS-Windows and DOS, as well as a command line compiler that runs under DOS. For this project, the Windows compilers were not used, but both the command line and the integrated (with an editor and debugger) DOS compilers were used.

After the initial setup, the Borland C++ package was used as shipped, with two exceptions. First, a Borland issued patch was applied to fix problems with the C++ new operator. This patch is BC31P1.ZIP. This patch was obtained from the Borland Forum on Compuserve, and included on the software diskette.

The second alteration to the standard compiler configuration was a modification in a header file that is used by the Turbo Vision library. This file, normally resides in the "borlandc\tvision\include" directory. It was altered in order to increase the safety pool size[2]. The modified file is available on the software diskette. After this file has been modified, the Turbo Vision library must be recompiled. This can take over 30 minutes, depending on the computational power and speed of the computer doing the processing.

In order to properly install and run the compiler package, the computer on which it is installed should have at least 4 Mb of RAM memory and at the very least 60 Mb of free disk space. The processor should be at least a very fast Intel 80386, but a 80486 is recommended.

TURBO VISION OVERVIEW

Turbo Vision is a set of libraries available for Borland's Pascal and C++ compilers. This library enables the programmer to create a graphically oriented, event driven interface with mouse support built in. This allows the programmer to move away from creating the event handling and screen manipulation software and focus on handling the data and other aspects of the system.

In the C++ implementation, the Turbo Vision library consists of a set of object oriented classes that can be used by the programmer. Information on Turbo Vision can be obtained from Borland, and from several books pertaining to Borland C++ or Turbo C++.

PROGRAM DESIGN - OVERVIEW

Both the Combine and FerryLog programs are written in C++ and conform to the AT&T 2.1 programing standard. Both programs make use of the standard C and C++ libraries, as well as the Turbo Vision Class library. Both programs make extensive use of the "new" and "destroy" (overloaded delete) operators to create and destroy objects

---

2 For more information about the safety pool, please see the Borland Turbo Vision documentation.

dynamically using the standard free store (heap). Neither program uses floating point mathematics, so floating point support can be left out during program linking in order to reduce the executable size of the final program.

FERRYLOG PROGRAM - BASIC PREMISE

The basic task of the FerryLog program is to prompt the bridge crew for pertinent information about the operation of their vessel using a friendly, graphical users interface. This information is then organized and recorded so that it can be transmitted to Colman Dock and incorporated into a database. In order to do this, the program uses a handy Turbo Vision object called a "collection." Turbo Vision collections are designed so that they are flexible in the types and amount of information that they contain. The FerryLog program uses collections by prompting for information using graphical elements, acquiring this information and organizing it using TObject derivatives dubbed "packets." This information is then passed to the collections, which hold a preset number of the packets before writing them to the disk and deleting them from memory. Before deletion, these packets are also written to a scrap file, "visual.dat" that allows the FerryLog user to see on the screen all the data that has been recorded during that session[3]. If the Print-As-You-Go option has been turned on by the user, then the collection will also send that information to the printer.

Once the user chooses to terminate the program (presumably at the end of a shift), the program will attempt to copy the data to a new file on the hard drive and on floppy drive 'A'. If the Summary print option has been turned on by the user, the FerryLog program will read the data file just created and print a formatted version of that data.

---

3 At the end of the session this file should be deleted, either by the FerryLog program or, as is currently the case, by a batch file that the FerryLog program runs within.

PROPOSED IMPROVEMENTS

In addition to the improvements detailed in Part One of this technical report, a general editing and reorganization of the program code is recommended. This reorganization should reduce the size of the source code and executable file sizes, as well as improve the readability of the code (which will allow easier program maintenance in the future). It would also facilitate future program enhancements to the FerryLog software.

A minor change recommended for the interface is the addition of another drill category to the current set (abandon ship, fire, and boat). Minor improvements could be made to many of the input screens in order to make data entry easier. Another improvement would be to link all the labels (TLabel objects) that are not currently linked with input lines (TInputline objects). At some point, context sensitive help should be integrated into the program. This could be done fairly easily by using the facilities built into Turbo Vision. The present implementation of persistent names for the "B" watch should be revised so that it encompasses all the vessel's watches.

At some point the heap amount viewer (used for debugging) should be removed from the program code, as this facility could be confusing to the end user. Further potential improvements include improving the ability of the program to check if the printer is not responding and report it to the user. Finally, when C++ compilers that conform to the AT&T 3.0 standard become available, implementation of a good exception handler using the try--throw--catch facility should be installed.

COMBINE PROGRAM - BASIC PREMISE

The main task of the Combine program is to take the data created by the FerryLog program on opposite ends of a vessel and combine them into one file that is readable by standard spreadsheet (or other analytical) programs. In order to accomplish this task, the Combine program also uses Turbo Vision collections. It first checks the names of the two input files to verify that they are a valid matching pair, and then begins to parse the data

contained in each. The objective of this parsing is to move the data from the files into collections so that it can be organized and merged together in a meaningful way.

After some initial parsing has been completed, the Combine program checks to see that the data in the two files is not too far apart temporally. (That is, the data from the two files are related to each other by time of day.) If the data are not too far apart, the program continues to parse and organize the data in each file. After all data has been parsed and organized, a new file is written with this data in a format that is easily readable. To facilitate readability, this file is a tab delimited text file, so that when it is read into a spreadsheet, each field will appear in its own cell.

COMBINE PROGRAM - IMPROVEMENTS

The Combine program was set up to write a spreadsheet readable text file because, as of this writing, it is not known which database program WSF operations staff will adopt in the future. Rbase is currently used to a limited extent in the WSF office, but some members of the WSF staff expect Sybase to be adopted at a later date. Due to this uncertainty, TRAC staff decided that the best approach to developing the prototype Combine program was to combine the files from the two wheel houses into one file that could be easily readable with a variety of analytical software programs. The most important improvement to the Combine program would be to change the way the program writes out information so that it could be more easily read into whatever database program WSF decides on as their standard. While delimited ASCII files can be input into most database programs, a more complex file structure could both speed this input, and allow a more effective translation to the database structure developed by WSF.

Another possible improvement to the Combine program would be to change the present character oriented command line interface into a graphical one. This could easily be accomplished by using the Turbo Vision tools that were employed in the FerryLog program.

APPENDIX A

PROGRAM GUIDE AND RELATED INFORMATION

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

BUFFERS.H

```
// safety pool size changed  EHB original file is buffers.hhh
// this is used by the turbo vision library programs and should
// reside in the \borlandc\tvision\include directory.

// after this file is placed in that directory, replacing the standard
// buffers.h file that came with Borland C++ 3.1 with Application Frameworks,
// the tvision library must be recompiled


/* ------------------------------------------------------*/
/*                                                       */
/*  BUFFERS.H                                            */
/*                                                       */
/*  Copyright (c) Borland International 1991             */
/*  All Rights Reserved.                                 */
/*                                                       */
/*  defines the functions getBufMem() and freeBufMem() for use   */
/*  in allocating and freeing viedo buffers              */
/*                                                       */
/* ------------------------------------------------------*/

#pragma option -Vo-
#if defined( __BCOPT__ )
#pragma option -po-
#endif

#if defined( Uses_TVMemMgr ) && !defined( __TVMemMgr )
#define __TVMemMgr

//const DEFAULT_SAFETY_POOL_SIZE = 4096;
const DEFAULT_SAFETY_POOL_SIZE = 32760;

class TBufListEntry
{

private:

   TBufListEntry( void*& );
   ~TBufListEntry();

   void *operator new( size_t, size_t );
   void *operator new( size_t );
   void operator delete( void * );

   TBufListEntry *next;
   TBufListEntry *prev;
   void*& owner;

   static TBufListEntry *near bufList;
   static Boolean freeHead();

   friend class TVMemMgr;
   friend void *operator new( size_t );

};
```

```
class TVMemMgr
{

public:

   TVMemMgr();

   static void resizeSafetyPool( size_t = DEFAULT_SAFETY_POOL_SIZE );
   static int safetyPoolExhausted();

   static void allocateDiscardable( void *&, size_t );
   static void freeDiscardable( void * );

private:

   static void * near safetyPool;
   static size_t near safetyPoolSize;
   static int near inited;
   static int initMemMgr();

};

#endif  // Uses_TVMemMgr

#pragma option -Vo.
#if defined( __BCOPT__ )
#pragma option -po.
#endif
```

C_FUNC2.CPP

```
// this is file c_func2.cpp  This file contains misc "C" programs
// for use with the ferrylog program

// It has a number of routines which could be improved upon

#include <errno.h>
#include <dos.h>
//#include "consts.h"
#include "frrydat2.h"
#include "utilfncs.h"
#include "misgui1.h"
#include <fstream.h>
#include <stdio.h>
#include <assert.h>
#include <io.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>

// sound function
extern prnstream PrinterStream;
extern PrintOptionData POD;
void MakeASound(int sound_type) {
        switch (sound_type){
                case sound_record:
                        sound (440);
                        delay (100);
                        nosound();
                break;
                case sound_oops:
                        sound(75);
                        delay (1200);
                        nosound();
                break;
                case sound_alert:
                        sound (275);
                        delay(500);
                        nosound();
                        sound(150);
                        delay(600);
                        nosound();
        }
}
const char near * VesselNameLookup (int  vessel_cmd) {

        /* vessel_cmd is the command number that was
        generated by pushing buttons in the vessel identification view */

        switch (vessel_cmd) {
                case id_ves_cath:
                        return VES_NAME_CATH;

                case id_ves_chel:
                        return VES_NAME_CHEL;

                case id_ves_elwh:
```

```
        return VES_NAME_ELWH;

case id_ves_eveg:
        return VES_NAME_EVEG;

case id_ves_hiyu:
        return VES_NAME_HIYU;

case id_ves_hyak:
        return VES_NAME_HYAK;

case id_ves_illa:
        return VES_NAME_ILLA;

case id_ves_issa:
        return VES_NAME_ISSA;

case id_ves_kala:
        return VES_NAME_KALA;

case id_ves_kale:
        return VES_NAME_KALE;

case id_ves_kits:
        return VES_NAME_KITS;

case id_ves_kitt:
        return VES_NAME_KITT;


case id_ves_klah:
        return VES_NAME_KLAH;

case id_ves_klic:
        return VES_NAME_KLIC;


case id_ves_nisq:
        return VES_NAME_NISQ;

case id_ves_olym:
        return VES_NAME_OLYM;

case id_ves_quin:
        return VES_NAME_QUIN;

case id_ves_rhod:
        return VES_NAME_RHOD;

case id_ves_seal:
        return VES_NAME_SEAL;

case id_ves_skag:
        return VES_NAME_SKAG;

case id_ves_spok:
        return VES_NAME_SPOK;

case id_ves_till:
```

```
                return VES_NAME_TILL;

            case id_ves_tyee:
                return VES_NAME_TYEE;


            case id_ves_wall:
                return VES_NAME_WALL;


            case id_ves_yaki:
                return VES_NAME_YAKI;


        }
        // return value here is invalid
        return "Vessel Naming_Error";
}
const char near * RouteNameLookup (int  route_cmd_number) {


// this function returns the text string corresponding to a
// route number generated by pressing the route init view's buttons

        switch (route_cmd_number) {
            case id_route_anac:
                return ROUTE_NAME_ANAC;
            case id_route_town:
                return ROUTE_NAME_TOWN;
            case id_route_muki:
                return ROUTE_NAME_MUKI;
            case id_route_edmd:
                return ROUTE_NAME_EDMD;
            case id_route_seaw:
                return ROUTE_NAME_SEAW;
            case id_route_seab:
                return ROUTE_NAME_SEAB;
            case id_route_fvso:
                return ROUTE_NAME_FVSO;
            case id_route_seav:
                return ROUTE_NAME_SEAV;
            case id_route_poit:
                return ROUTE_NAME_POIT;
        }
        // return here is invalid

        return "Route Naming Error";
}

// function to return the names of the files to contain the data

FerryFileNames DataFileNames (void) {

        time_t temp_time;
        temp_time=time(NULL);
        tm *temp_tm =localtime(&temp_time);
        char hour[3];
        char month[3];
        char day[3];
```

```
        char year[3];
        char ves_string[4];
        FerryFileNames return_names;
        int hour_int;

        strftime(hour,3,"%H", temp_tm);
        hour_int=atoi(hour);
        strftime(month,3,"%m", temp_tm);
        strftime(day, 3,"%d", temp_tm);
        strftime(year, 3,"%y", temp_tm);

// now read in the data from the hidden registration file

        FILE *reg_file;
        if( (reg_file=fopen("ferryid.dat","r"))==NULL) {
                strcpy(ves_string, "UK9\0");
        }
        else {
                fgets (ves_string, 4,reg_file);
                fclose (reg_file);
        }


// now put the string elements where appropriate
        return_names.FinalFileName[0]=month[0];
        return_names.FinalFileName[1]=month[1];
        return_names.FinalFileName[2]=day[0];
        return_names.FinalFileName[3]=day[1];
        return_names.FinalFileName[4]=year[0];
        return_names.FinalFileName[5]=year[1];
        // calculate the appropriate code for the hour based on the alphabet
        char letter;
        letter =hour_int+65;  // 65 is ascii code for 'A'
                              // this should work

        return_names.FinalFileName[6]=letter;
        return_names.FinalFileName[7]=ves_string[2];
        return_names.FinalFileName[8]='.';
        return_names.FinalFileName[9]=ves_string[0];
        return_names.FinalFileName[10]=ves_string[1];
        return_names.FinalFileName[11]='F';
        return_names.FinalFileName[12]='\0';
        strncpy(return_names.TempFileName, return_names.FinalFileName,
                13);
        return_names.TempFileName[11]='T';
        return_names.TempFileName[12]='\0';

        return return_names;
}

int DataFileCheck (FerryFileNames origname, char filetype, int rep_number) {

        // filetype is either 'T' for temp or 'F' for final.
        // anything else will cause program to abort
        const int CHANGE=1;
        const int NOCHANGE=0;
        int recursive_return_value=0;

        assert(filetype=='T' || filetype=='F');
```

```
FerryFileNames newname=origname;

// check that rep_number >=0 <=9
if (rep_number < 0) rep_number=0;
if (rep_number >9) rep_number=9;

int access_result;
 // check for read access
// check temporary file
if (filetype=='T') { // temporary file check
        access_result=access(origname.TempFileName, 04);
        if (access_result==0 ||
                (access_result==(-1) && errno==EACCES)) {
                //file exists, and we need to
                //rename the original file name
                // before we write the new file

                // alter the name of the file

                newname.TempFileName[6]=rep_number+48;
                // ascii code for '0'
                newname.TempFileName[11]='U';  //code for temp oops
                // now start recursion process
                rep_number++;
                recursive_return_value=
                        DataFileCheck(newname, filetype, rep_number);
                // return if change performed
                if (recursive_return_value==CHANGE) return CHANGE;

                // " rename" by using mycopy to copy the file,
                // then delete the old file
                mycopy (origname.TempFileName, newname.TempFileName);

                char system_string[22];
                strncpy (system_string, " ",22);
                strcpy (system_string, "Del ");
                strcat(system_string, origname.TempFileName);
                system(system_string);
                return CHANGE;
        }
}

if (filetype=='F') { // Final file check
        access_result=access(origname.FinalFileName, 04);
        if (access_result==0 ||
        (access_result==(-1) && errno==EACCES)) {
        //file exists, and we need to
        //rename the original file name
        // before we write the new file

        // alter the name of the file
        // in the case of duplicate final file names,
        // we should also change the name of the temporary file

        newname.TempFileName[6]=rep_number+48;
        // ascii code for '0'
        newname.TempFileName[11]='U';  //code for temp oops
```

```
                newname.FinalFileName[6]=rep_number+48;
                // ascii code for '0'
                newname.FinalFileName[11]='G';  //code for final oops
                rep_number++;
                recursive_return_value=
                        DataFileCheck(newname, filetype, rep_number);
                // return if change performed
                if (recursive_return_value==CHANGE) return CHANGE;

                // " rename" by using mycopy to copy the file,
                // then delete the old file


                mycopy (origname.FinalFileName, newname.FinalFileName);

                char system_string[22];
                strncpy (system_string, " ",22);
                strcpy (system_string, "del ");
                strcat(system_string, origname.FinalFileName);
         system(system_string);
                return CHANGE;
                }
        }
        return NOCHANGE;
}
void SpewPrintOn(void) {

        PrinterStream.init();  // initialize printer, eject page

}
void SpewPrintOff(void) {
        POD.spewData=Off;
        //PrinterStream << "\f"; // form feed
        //PrinterStream.flush();

}
void SummaryPrintOn(void) {
        // do nothing yet
        //PrinterStream.init();  // initialize printer, eject page

}
void SummaryPrintOff(void) {
        POD.finalData=Off;
        //PrinterStream << "\f"; // form feed
        //PrinterStream.flush();

}

void WatchCrewLoad(CrewInfo *crew_ptr) {
        ifstream data;

        if (strcmp(crew_ptr->watch_name,ID_WATCH_B)==0) {
                data.open("crewdat.b",ios::in);
                if (!data) return;
                data.getline(crew_ptr->crew_name[0],crew_name_length);
                data.getline(crew_ptr->crew_name[1],crew_name_length);
                data.getline(crew_ptr->crew_name[2],crew_name_length);
                data.getline(crew_ptr->crew_name[3],crew_name_length);
```

```
                data.getline(crew_ptr->crew_name[4],crew_name_length);
                data.getline(crew_ptr->crew_name[5],crew_name_length);
                data.getline(crew_ptr->crew_name[6],crew_name_length);
                data.getline(crew_ptr->crew_name[7],crew_name_length);
                data.getline(crew_ptr->crew_name[8],crew_name_length);
                data.getline(crew_ptr->crew_name[9],crew_name_length);
                data.getline(crew_ptr->crew_name[10],crew_name_length);
                data.close();
        }
}

void WatchCrewSave(CrewInfo *crew_ptr) {
        ofstream data;

        if (strcmp(crew_ptr->watch_name,ID_WATCH_B)==0) {
                data.open("crewdat.b",ios::out);
                if (!data) return;
                data <<crew_ptr->crew_name[0]<<"\n";
                data <<crew_ptr->crew_name[1]<<"\n";
                data <<crew_ptr->crew_name[2]<<"\n";
                data <<crew_ptr->crew_name[3]<<"\n";
                data <<crew_ptr->crew_name[4]<<"\n";
                data <<crew_ptr->crew_name[5]<<"\n";
                data <<crew_ptr->crew_name[6]<<"\n";
                data <<crew_ptr->crew_name[7]<<"\n";
                data <<crew_ptr->crew_name[8]<<"\n";
                data <<crew_ptr->crew_name[9]<<"\n";
                data <<crew_ptr->crew_name[10]<<"\n";
                data.close();
        }
}
```

C_FUNC2.H

```
// this file is c_func2.h
// it contains function declarations for the c_func2.cpp file

void SpewPrintOff (void);
void SpewPrintOn (void);
void SummaryPrintOn(void);
void SummaryPrintOff(void);
```

C_FUNCR.CPP

```
// this file is C_funcr.cpp


/* replacement functions for standard library functions that don't work right
These were written to replace standard library functions that weren't
working right under the conditions that I was trying to use them

*/
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char * mctime (time_t * time) {
        char temptime[26];
        strcpy (temptime, ctime(time));
        temptime[24]='\0';  // remove new line
        return temptime;
}

void mstrcpy (char *s, char *t) {
        while ((*s=*t )!='\0') {
                s++;
                t++;
        }
}

void mstrncpy (char *s, char *t, int n) {
        int counter=0;
        int t_size =strlen(t);

        for (counter=0; counter < n; counter++) {
                s[counter]=t[counter];
                if (t[counter]=='\0') break;
        }

        for(counter++; counter <n; counter++) {
                s[counter]='\0';
        }
}
```

C_FUNCR.H

```
// this file is c_funcr.h

/* header file for c function replacement functions */
#include <time.h>
/* replaces strcpy */
void mstrcpy (char *s, char *t);
/* replaces strncpy */
char * mctime (time_t * time);
void mstrncpy (char *s, char *t, int n);
```

COMBDAT.CPP

```
// this file is combdat.cpp

// this file is used to build the combine program as well as the ferrylog
//      Program.
/* this file defines the way collections are used in the combine program,
and contains information to make insertions into the collection
typesafe */

#define Uses_TNSSortedCollection
#include "frrydat2.h"
#include "combdat.h"
#include <time.h>
#include <tv.h>

CombINITDataCollection::CombINITDataCollection
        (ccIndex aLimit, ccIndex aDelta) :
        TNSSortedCollection(aLimit, aDelta) {
        // maybe add something?
}
/*
void CombINITDataCollection::writeAll(void) {
        forEach(callWrite, &fos //*file pointer here  );
        //        fos << "marker \n";  // remove later for testing only
        fos << flush;
}
*/
void * CombINITDataCollection::keyOf(void *item) {

        DataInitializationDataPacket * DIDP;
        DIDP = (DataInitializationDataPacket *) item;
        return (void *) &DIDP->initialize_time.givenTime;


}
int  CombINITDataCollection::compare (void *key1,void *key2) {

        time_t *test1=0, *test2=0;
        test1=(time_t*)key1;
        test2=(time_t*)key2;

        if (*test1 < *test2) return (-1);
        if (*test1== *test2) return (0);
        if (*test1 > *test2) return (1);
}

ccIndex CombINITDataCollection::insert (DataInitializationDataPacket *DIDP) {

        return TNSSortedCollection::insert(DIDP);
}
// movement

CombMVDataCollection::CombMVDataCollection(ccIndex aLimit, ccIndex aDelta) :
                TNSSortedCollection(aLimit, aDelta) {
                // maybe add something?
}
/*
```

```
void CombMVDataCollection::writeAll(void) {
        forEach(callWrite, &fos //file pointer here  );
        fos << flush;
}
*/
void * CombMVDataCollection::keyOf(void *item) {

        MovementDataPacket * MDP;
        MDP = (MovementDataPacket *) item;
        return (void *) &MDP->time_data.givenTime;



}
int  CombMVDataCollection::compare (void *key1,         void *key2) {

        time_t *test1=0, *test2=0;
        test1=(time_t*)key1;
        test2=(time_t*)key2;

        if (*test1 < *test2) return (-1);
        if (*test1== *test2) return (0);
        if (*test1 > *test2) return (1);
}

ccIndex CombMVDataCollection::insert (MovementDataPacket *MDP) {

        return TNSSortedCollection::insert(MDP);
}

// Drills


CombDRDataCollection::CombDRDataCollection(ccIndex aLimit, ccIndex aDelta) :
                TNSSortedCollection(aLimit, aDelta) {
                // maybe add something?
}
/*
void CombDRDataCollection::writeAll(void) {
        forEach(callWrite, &fos //file pointer here  );
        //        fos << "marker \n";  // remove later for testing only
        fos << flush;
}
*/
void * CombDRDataCollection::keyOf(void *item) {

        DrillsDataPacket * DDP;
        DDP = (DrillsDataPacket *) item;
        return (void *) &DDP->time_data.givenTime;

}
int  CombDRDataCollection::compare (void *key1,void *key2) {

        time_t *test1=0, *test2=0;
        test1=(time_t*)key1;
        test2=(time_t*)key2;

        if (*test1 < *test2) return (-1);
        if (*test1== *test2) return (0);
```

```
        if (*test1 > *test2) return (1);

}

ccIndex CombDRDataCollection::insert (DrillsDataPacket *DDP) {

        return TNSSortedCollection::insert(DDP);
}

// incidents


CombINCDataCollection::CombINCDataCollection(ccIndex aLimit, ccIndex aDelta) :
                TNSSortedCollection(aLimit, aDelta) {
                // maybe add something?
}
/*
void CombINCDataCollection::writeAll(void) {
        forEach(callWrite, &fos //file pointer here  );
        //      fos << "marker \n";  // remove later for testing only
        fos << flush;
}
*/
void * CombINCDataCollection::keyOf(void *item) {

        IncidentsDataPacket * IDP;
        IDP = (IncidentsDataPacket *) item;
        return (void *) &IDP->time_data.givenTime;

}
int  CombINCDataCollection::compare (void *key1,void *key2) {

        time_t *test1=0, *test2=0;
        test1=(time_t*)key1;
        test2=(time_t*)key2;

        if (*test1 < *test2) return (-1);
        if (*test1== *test2) return (0);
        if (*test1 > *test2) return (1);

}

ccIndex CombINCDataCollection::insert (IncidentsDataPacket *IDP) {

        return TNSSortedCollection::insert(IDP);
}

// weather


CombWxDataCollection::CombWxDataCollection(ccIndex aLimit, ccIndex aDelta) :
                TNSSortedCollection(aLimit, aDelta) {
                // maybe add something?
}

void * CombWxDataCollection::keyOf(void *item) {

        WeatherDataPacket * WDP;
        WDP = (WeatherDataPacket *) item;
```

```
        return (void *) &WDP->weatherTime.givenTime;


}
int  CombWxDataCollection::compare (void *key1,         void *key2) {

        time_t *test1=0, *test2=0;
        test1=(time_t*)key1;
        test2=(time_t*)key2;

        if (*test1 < *test2) return (-1);
        if (*test1== *test2) return (0);
        if (*test1 > *test2) return (1);
}

ccIndex CombWxDataCollection::insert (WeatherDataPacket *WDP) {

        return TNSSortedCollection::insert(WDP);
}

#ifdef COMBINE
        // don't define if part of ferrylog program
void callWrite(void *FDP, void *FOF /* file pointer here */) {

        ((FerryDataPacket *)FDP)->write((ofstream *) FOF);
        // call the appropriate write function
}
#endif
```

COMBDAT.H

```
// this file is combdat.h
/* this file contains information useful to the combine and ferrylog
programs and contains class declarations for TVision collections
This helps with type safeness
*/

#define Uses_TNSSortedCollection
#include "frrydat2.h"
#include <tv.h>

// TNSSortedCollection class
class CombINITDataCollection: public TNSSortedCollection {
public:
        CombINITDataCollection(ccIndex aLimit, ccIndex aDelta) ;
        virtual void *keyOf(void  *item);
        ccIndex insert (DataInitializationDataPacket *DIDP);
   //   void writeAll (void);
        ofstream fos;
private:
        virtual int compare (void *key1, void *key2);

};
class CombINCDataCollection: public TNSSortedCollection {
public:
        CombINCDataCollection(ccIndex aLimit, ccIndex aDelta) ;
        virtual void *keyOf(void *item);

        ccIndex insert (IncidentsDataPacket *IDP);
  //    void writeAll (void);
        ofstream fos;
private:
        virtual int compare (void  *key1, void *key2);

};

class CombDRDataCollection: public TNSSortedCollection {
public:
        CombDRDataCollection(ccIndex aLimit, ccIndex aDelta) ;
        virtual void *keyOf(void *item);
        ccIndex insert (DrillsDataPacket *DDP);
  //    void writeAll (void);
        ofstream fos;
private:
        virtual int compare (void  *key1,void *key2);
};
class CombMVDataCollection: public TNSSortedCollection {
public:
        CombMVDataCollection(ccIndex aLimit, ccIndex aDelta);
        virtual void *keyOf(void *);
        ccIndex insert (MovementDataPacket *MDP);
  //    void writeAll (void);
        ofstream fos;
private:
        virtual int compare (void *key1, void *key2);
};
```

```
// weather
class CombWxDataCollection: public TNSSortedCollection {
public:
        CombWxDataCollection(ccIndex aLimit, ccIndex aDelta);
        virtual void *keyOf(void *);
        ccIndex insert (WeatherDataPacket *WDP);
   //   void writeAll (void);
        ofstream fos;
private:
        virtual int compare (void *key1, void *key2);
};

void callWrite( void *FDP, void *FOF );  // has to not be a class member function
// FerryDataPacket is FDP type
//FOF is Ferry Output File  and is a pointer to an ofstream
```

CMBINE.BAT

```
rem combine batch file

combine %1 %2 %3 > combout.out
type combout.out
del combout.out
```

COMBINE.CPP

```
/* this program works to combine the two files associated with a
particular vessel-shift for the Washington State Ferries */
//. this file is combine.cpp and is the main file for the combine program

// main program

/* problem with string overrun is in the function HandleIncident */

#include "c_funcr.h"
#include <assert.h>
#include <stdio.h>
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>
#include "consts.h"
#include "frrydat2.h"
#include "combdat.h"
#include <dos.h>
#include <time.h>
// #pragma intrinsic -strncpy
unsigned _stklen=8192;
#include <tv.h>
const int MAXDATALINE =400;
const int MAXOUTLINELENGTH=500;
const int MAXCOMMANDLENGTH=50;
const int TRUE = 1;
const int FALSE= 0;
const int MaxTimeDiff=(24) *(60 *60); // maximum time diff in hours
struct {
        FILE *infile1;
        FILE *infile2;
        FILE *outfile;
} typedef file_ptrs;
// string mini class

void code_strip(char *return_code, char *file);
int HandleInitialize(char data_line[MAXDATALINE], int);
int HandleMovement (char data_line[MAXDATALINE], int);
int HandleIncident (char data_line[MAXDATALINE], int);
int HandleDrills (char data_line[MAXDATALINE], int);
int HandleWeather (char data_line[MAXDATALINE], int);
int BottomCheck(FILE *);

/* global variables */
CombINITDataCollection collection_init[2]= {CombINITDataCollection(2,2),
                CombINITDataCollection(2,2)};
CombMVDataCollection collection_move[2]={CombMVDataCollection(30,5),
        CombMVDataCollection(30,5)};
CombDRDataCollection collection_drills[2]={CombDRDataCollection(5,2),
        CombDRDataCollection(5,2)};
CombINCDataCollection collection_incid[2]={CombINCDataCollection(20,3),
        CombINCDataCollection(20,3)};
CombWxDataCollection collection_weath[2]={CombWxDataCollection(10,3),
        CombWxDataCollection(10,3)};
```

```
ofstream fos;  // output file

void main (int argc, char *argv[]) {
        file_ptrs in_out_files;

        // check argument numbers

        if (argc!=4) {
                fprintf(stdout, "Usage: frrycomb file1 file2 outfile \n");
                fprintf(stdout, "This programs combines two input files");
                fprintf(stdout, " from each end of the vessel \n");
                fprintf(stdout, "and creates a file viewable by excel\n");
                exit (-1);
        }
        // check output file for existence
        int File_is_check=access(argv[3], 0);
        if (File_is_check==0) {  // file already exists
                fprintf(stdout, "Sorry, output file %s exists. ", argv[3]);
                fprintf(stdout,"Can't overwrite. \n ");
                exit (-1);
        }

        // now open the input files
        // infile1

         if ( (in_out_files.infile1=fopen(argv[1], "r"))==NULL) {
                fprintf(stdout, "Sorry, can't open file %s. \n",argv[1]);
                exit (-1);
        }

        //infile2
        if ( (in_out_files.infile2=fopen(argv[2], "r"))==NULL) {
                fprintf(stdout, "Sorry, can't open file %s. \n",argv[2]);
                exit (-1);
        }
        // now check that infiles are from the same vessel
        // check the file names against each other
        char vesselcode1[4], vesselcode2[4];
        vesselcode1[3]='\0'; vesselcode2[3]='\0';

        // vesselcode1=code_strip(argv[1]);
        code_strip(vesselcode1, argv[1]);
        code_strip(vesselcode2, argv[2]);
        if (strncmp(vesselcode1, vesselcode2,2) !=0) { // compare name
                fprintf(stdout, "Files not from the same Ferry! \n");
                fprintf(stdout, "File Names don't match properly!\n");
                fprintf(stdout, "Program terminating \n");
                fcloseall();
                exit (-1);
        }
        if (strncmp(vesselcode1, vesselcode2,3) ==0) { // compare name
                // if true, same end
                fprintf(stdout, "Files from the same vessel end! \n");
                fprintf(stdout, "File Names don't match properly!\n");
                fprintf(stdout, "Program terminating \n");
                fcloseall();
                exit (-1);
        }
```

```
if (vesselcode1[2]!='1') {
        fprintf(stdout, "%s is not from end one! \n", argv[1]);
        fprintf(stdout, "File Names don't match properly!\n");
        fprintf(stdout, "Program terminating \n");
}
if (vesselcode2[2]!='2') {
        fprintf(stdout, "%s is not from end two! \n", argv[2]);
        fprintf(stdout, "File Names don't match properly!\n");
        fprintf(stdout, "Program terminating \n");
}

// check each input file for special file terminator
if (BottomCheck(in_out_files.infile1)) {
        fprintf(stdout, "%s is not properly terminated. ",argv[1]);
        fprintf(stdout, "Some data might be lost! \n");
}
rewind(in_out_files.infile1);

if (BottomCheck(in_out_files.infile2)) {
        fprintf(stdout, "%s is not properly terminated. ",argv[2]);
        fprintf(stdout, "Some data might be lost! \n");
}
rewind(in_out_files.infile2);

// now start parsing and loading the data

// do file number one

char read_line[MAXDATALINE];
char pass_line[MAXDATALINE];
strncpy(read_line, " ",MAXDATALINE);
strncpy(pass_line, " ",MAXDATALINE);

char *token;
while (  fgets (read_line, MAXDATALINE, in_out_files.infile1)!=NULL) {
        //check string and then hand off information to
        //appropriate subfunction
        int result=0;
        strncpy(pass_line,read_line,MAXDATALINE);
        token=strtok(read_line, "\t");
        if (strcmp (token, FILE_MARKER_BOTTOM)==0); // do nothing

        else if (strcmp (token, FILE_MARKER_TOP)==0); // do nothing
        else if (strcmp (token, ACTION_INITIALIZE)==0){
                result=HandleInitialize(pass_line, 1); /*
                        1 in call indicates infile one */
        }

        else if (strcmp (token, ACTION_MOVEMENT) ==0) {
                result=HandleMovement(pass_line, 1);
        }
        else if (strcmp (token, ACTION_INCIDENTS)==0) {
                result=HandleIncident(pass_line, 1);
        }
        else if (strcmp (token, ACTION_DRILLS)==0) {
                result=HandleDrills(pass_line, 1);
        }
        else if (strcmp(token, ACTION_WEATHER)==0) {
                result=HandleWeather(pass_line,1);
```

```
        }
}
while (  fgets (read_line, MAXDATALINE, in_out_files.infile2)!=NULL) {
        //check string and then hand off information to
        //appropriate subfunction
        int result=0;
        strncpy(pass_line,read_line,MAXDATALINE);
        token=strtok(read_line, "\t");
        if (strcmp (token, FILE_MARKER_BOTTOM)==0); // do nothing

        else if (strcmp (token, FILE_MARKER_TOP)==0); // do nothing
        else if (strcmp (token, ACTION_INITIALIZE)==0){
                result=HandleInitialize(pass_line, 2); /*
                        1 in call indicates infile one */
        }

        else if (strcmp (token, ACTION_MOVEMENT) ==0) {
                result=HandleMovement(pass_line, 2);
        }
        else if (strcmp (token, ACTION_INCIDENTS)==0) {
                result=HandleIncident(pass_line, 2);
        }
        else if (strcmp (token, ACTION_DRILLS)==0) {
                result=HandleDrills(pass_line, 2);
        }
        else if (strcmp(token, ACTION_WEATHER)==0) {
                result=HandleWeather(pass_line,2);
        }
}

fcloseall();

// now test the files times for reasonably close adherence

DataInitializationDataPacket *DID1=
        (DataInitializationDataPacket *) collection_init[0].at(0);
DataInitializationDataPacket *DID2=
        (DataInitializationDataPacket *) collection_init[1].at(0);

time_t difftime;
difftime= labs(  (DID1->initialize_time.machineTime)
        -(DID2->initialize_time.machineTime)  );

if (difftime > MaxTimeDiff) {
        fprintf(stdout,
                "Time difference between files is too great \n");
        exit(-1);
}
// check for same route
if (strcmp (DID1->routeInit.route_name,
        DID2->routeInit.route_name)!=0){
        fprintf(stdout, "Ends are not running the same route! \n");
        exit (-1);
}

time_t time_now;
time_now=time(NULL);

char stime1[26],stime2[26], nowtime[26];
```

```
strncpy(stime1,ctime(&DID1->initialize_time.machineTime),26 );
stime1[24]=' ';
strncpy(stime2,ctime(&DID2->initialize_time.machineTime),26);
strncpy(nowtime,ctime(&time_now),26);


fos.open(argv[3],ios::out);
fos << "Combining report \n\n"
<< "Vessel:\t"<<vesselcode1[0]<<vesselcode1[1]
<<"\t" <<"\n" << "Creation Date:\t\t"
<<"End one:\tEnd two:\n"
<<"\t\t" <<stime1 <<"\t"
<<stime2 <<"\n\n"
<<"Files:\t\t" << argv[1] << "\t"<<argv[2] <<"\n"
<<"Combined on:\t" << nowtime <<"\n\n";
fos << "Initial Route:\t" << DID1->routeInit.route_name << "\t\n"
<<"End one running: \t" << "Arrival: \t" <<DID1->moveInit.arrival
<<"\t\t"
<< "Departure:\t"<< DID1->moveInit.depart <<"\t\n"
<<"End two running: \t" << "Arrival: \t"
<<DID2->moveInit.arrival <<"\t\t"
<< "Departure:\t"<< DID2->moveInit.depart <<"\t\n\n";

fos << "End One Information: \n\n";
DID1->ExcPrint(&fos); // print the rest of the init data
fos <<"\n" << "End Two Information: \n\n";
DID2->ExcPrint(&fos); // print the rest of the init data

fos << "\n\n";
//now print the movement data for end one

fos<<"Please note: Time information below is in the following format:"
<<"\n" <<"DD:HH:MM:SS\n"
<<"\tWhere DD equals day of the month;\n"
<<"\tHH equals hour of the day;\n"
<<"\tMM equals minute of the hour;\n"
<<"\tSS equals second of the minute.\n";
fos <<"\n\n";
fos <<"End one TimeTable \n";

ccIndex Mcount=0; int move_counter=0;
char Movelines[4][MAXOUTLINELENGTH];
char temp[30];
strncpy (temp, " ", 30);
strncpy(Movelines[0]," ",MAXOUTLINELENGTH);
strncpy(Movelines[1]," ",MAXOUTLINELENGTH);
strncpy(Movelines[2]," ",MAXOUTLINELENGTH);
strncpy(Movelines[3]," ",MAXOUTLINELENGTH);

strcat (Movelines[0], "Type\t");
strcat (Movelines[1], "Name\t");
strcat (Movelines[2], "M Time\t");
strcat (Movelines[3], "G Time\t");

Mcount=collection_move[0].getCount();
MovementDataPacket *MDP =NULL;

for (move_counter=0; move_counter < Mcount; move_counter++) {
```

```
        MDP = (MovementDataPacket *)
                collection_move[0].at(move_counter);
        if ((strcmp(MDP->point_type,ACTION_DEPART)==0)
                && (move_counter!=0 ) ) {
                fos <<  Movelines[0]<<"\n"
                <<Movelines[1] <<"\n"
                << Movelines[2]<<"\n"
                <<Movelines[3] <<"\n\n";
                for (int a_count=0; a_count < 4; a_count++) {
                        strncpy(Movelines[a_count]," ",
                        MAXOUTLINELENGTH);
                }
                strcat (Movelines[0], "Type\t");
                strcat (Movelines[1], "Name\t");
                strcat (Movelines[2], "M Time\t");
                strcat (Movelines[3], "G Time\t");
        }
        strcat(Movelines[0], MDP->point_type);
        strcat(Movelines[0],"\t");
        strcat(Movelines[1],MDP->name);
        strcat(Movelines[1],"\t");
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&MDP->time_data.machineTime));
        strcat(Movelines[2],temp);
        strcat(Movelines[2],"\t");
        strftime(temp, 30, " %d:%H:%M:%S",
                localtime(&MDP->time_data.givenTime));
        strcat(Movelines[3],temp);
        strcat(Movelines[3],"\t");
}
fos << Movelines[0]<<"\n"<<Movelines[1] <<"\n"
                << Movelines[2]<<"\n"<<Movelines[3] <<"\n\n";

fos <<"End two TimeTable \n\n";
strncpy(Movelines[0]," ",MAXOUTLINELENGTH);
strncpy(Movelines[1]," ",MAXOUTLINELENGTH);
strncpy(Movelines[2]," ",MAXOUTLINELENGTH);
strncpy(Movelines[3]," ",MAXOUTLINELENGTH);

strcat (Movelines[0], "Type\t");
strcat (Movelines[1], "Name\t");
strcat (Movelines[2], "M Time\t");
strcat (Movelines[3], "G Time\t");


Mcount=collection_move[1].getCount();

for (move_counter=0; move_counter < Mcount; move_counter++) {

        MDP = (MovementDataPacket *)
                collection_move[1].at(move_counter);
        if ( (strcmp(MDP->point_type,ACTION_DEPART)==0)
                && ( move_counter!=0) ) {
                fos <<  Movelines[0]<<"\n"
                <<Movelines[1] <<"\n"
                << Movelines[2]<<"\n"
                <<Movelines[3] <<"\n\n";
                for (int a_count=0; a_count < 4; a_count++) {
                        strncpy(Movelines[a_count]," ",
```

```
                        MAXOUTLINELENGTH);
                }
                strcat (Movelines[0], "Type\t");
                strcat (Movelines[1], "Name\t");
                strcat (Movelines[2], "M Time\t");
                strcat (Movelines[3], "G Time\t");
        }

        strcat(Movelines[0], MDP->point_type);
        strcat(Movelines[0],"\t");
        strcat(Movelines[1],MDP->name);
        strcat(Movelines[1],"\t");
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&MDP->time_data.machineTime));
        strcat(Movelines[2],temp);
        strcat(Movelines[2],"\t");
        strftime(temp, 30, " %d:%H:%M:%S",
                localtime(&MDP->time_data.givenTime));
        strcat(Movelines[3],temp);
        strcat(Movelines[3],"\t");


}
fos << Movelines[0]<<"\n"<<Movelines[1] <<"\n"
                << Movelines[2]<<"\n"<<Movelines[3] <<"\n\n";

fos << "End of Timetable Data \n\n";

// now put out incident information
// end one

fos << "Incident information End one: \n \n";
fos << "Incident:\tM Time\tG Time\t Comment \n";
int Icount = collection_incid[0].getCount(), icounter;
for (icounter=0; icounter <Icount; icounter++) {
        IncidentsDataPacket * IDP =(IncidentsDataPacket *)
                collection_incid[0].at(icounter);

        fos <<IDP->Incident_name <<"\t";
        strncpy(temp," ",30);
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&IDP->time_data.machineTime));
        fos << temp<<"\t";
        strncpy(temp," ",30);
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&IDP->time_data.givenTime));
        fos << temp<< "\t"
        << IDP->comment <<"\n";
}

// incidents end #2
fos << "\n Incident information End Two \n\n";

Icount = collection_incid[1].getCount();
for (icounter=0; icounter <Icount; icounter++) {
        IncidentsDataPacket * IDP =(IncidentsDataPacket *)
                collection_incid[1].at(icounter);

        fos <<IDP->Incident_name <<"\t";
```

```
        strncpy(temp," ",30);
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&IDP->time_data.machineTime));
        fos << temp<<"\t";
        strncpy(temp," ",30);
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&IDP->time_data.givenTime));
        fos << temp<< "\t"
        << IDP->comment <<"\n";


}

 // drills information
fos <<"\n\n" << "Drills information for End One \n";
fos << "drill:\tM Time\tG Time\t Comment \n";
int Dcount = collection_drills[0].getCount(), dcounter;
for (dcounter=0; dcounter <Dcount; dcounter++) {
        DrillsDataPacket * DDP =(DrillsDataPacket *)
                collection_drills[0].at(dcounter);

        fos <<DDP->drill_name <<"\t";
        strncpy(temp," ",30);
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&DDP->time_data.machineTime));
        fos << temp<<"\t";
        strncpy(temp," ",30);
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&DDP->time_data.givenTime));
        fos << temp<<"\t"
        << DDP->comment <<"\n";

}
fos <<"\n";

fos <<"\n\n" << "Drills information for End Two \n";
fos << "drill:\tM Time\tG Time\t Comment \n";
Dcount = collection_drills[1].getCount(), dcounter;
for (dcounter=0; dcounter <Dcount; dcounter++) {
        DrillsDataPacket * DDP =(DrillsDataPacket *)
                collection_drills[1].at(dcounter);

        fos <<DDP->drill_name <<"\t";
        strncpy(temp," ",30);
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&DDP->time_data.machineTime));
        fos << temp<<"\t";
        strncpy(temp," ",30);
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&DDP->time_data.givenTime));
        fos << temp<< "\t"
        << DDP->comment <<"\n";

}

// weather information

fos <<"\n\n" << "Weather information for End One \n";
fos <<"Time:\tClouds:\tPrecip:\tVisibility\tWind Direction:\t"
```

```
<< "Wind Speed:\tSea Condition:\tTemperature:\tBarometer\t:\n";


int Wcount = collection_weath[0].getCount(), wcounter;
for (wcounter=0; wcounter <Wcount; wcounter++) {
        WeatherDataPacket * WDP =(WeatherDataPacket *)
                collection_weath[0].at(wcounter);

        strncpy(temp," ",30);
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&WDP->weatherTime.machineTime));
        fos << temp<<"\t";
        fos << WDP->weatherData.Clouds <<"\t"
        <<WDP->weatherData.Precip <<"\t"
        <<WDP->weatherData.Vis <<"\t"
        <<WDP->weatherData.WindDir<<"\t"
        <<WDP->weatherData.WindSpeed<<"\t"
        <<WDP->weatherData.Sea<<"\t"
        <<WDP->weatherData.Temp<<"\t"
        <<WDP->weatherData.Barom<<"\n";

}
fos <<"\n";

// end two

fos <<"\n\n" << "Weather information for End Two \n";
fos <<"Time:\tClouds:\tPrecip:\tVisibility\tWind Direction:\t"
<< "Wind Speed:\tSea Condition:\tTemperature:\tBarometer\t:\n";


Wcount = collection_weath[1].getCount();
for (wcounter=0; wcounter <Wcount; wcounter++) {
        WeatherDataPacket * WDP =(WeatherDataPacket *)
                collection_weath[1].at(wcounter);

        strncpy(temp," ",30);
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&WDP->weatherTime.machineTime));
        fos << temp<<"\t";
        fos << WDP->weatherData.Clouds <<"\t"
        <<WDP->weatherData.Precip <<"\t"
        <<WDP->weatherData.Vis <<"\t"
        <<WDP->weatherData.WindDir<<"\t"
        <<WDP->weatherData.WindSpeed<<"\t"
        <<WDP->weatherData.Sea<<"\t"
        <<WDP->weatherData.Temp<<"\t"
        <<WDP->weatherData.Barom<<"\n";

}
fos <<"\n";


// end of weather information
// end of report

fos << "\n\nEnd of report \n";
fos.close();
fprintf(stdout,"Normal completion \n");
```

```
        return;


}
void code_strip(char *return_code, char *file){
        // reconstucts vessel code from filename
        return_code[3]='\0';
        return_code[0]=file[9];
        return_code[1]=file[10];
        return_code[2]=file[7];

}

int BottomCheck(FILE *infile) {
        int FileCmpFlag=TRUE;

        // returns False if special file terminator is found
        // true if otherwise
        char data_line[MAXDATALINE];
        char test_string[MAXCOMMANDLENGTH];
        char *chptr;
        while (  fgets (data_line, MAXDATALINE, infile)!=NULL) {
                chptr=strtok(data_line, "\t");
                if (strncmp (chptr, FILE_MARKER_BOTTOM,
                        strlen(FILE_MARKER_BOTTOM) )==0)
                        FileCmpFlag=FALSE;

        }
        return FileCmpFlag;
}

int HandleInitialize(char data_line[MAXDATALINE], int which_file){
        static DataStorageClass InitStuff;
        static OccurTime InitTime;
        static int crew_counter, tides_counter;

        char *tokenA, *tokenB, *tokenC ;
        tokenA=strtok(data_line, "\t");
        tokenB=strtok(NULL, "\t");  //must take second token
        if (strcmp(tokenB, INIT_KEYWD_VES_NAME)==0){
                tokenC=strtok(NULL, "\t");
                strncpy(InitStuff.vessel_data.vessel_name, tokenC,
                        MaxVesselNameLength);
                tokenC=strtok(NULL, "\t");
                time_t  time=atol(tokenC);
                InitTime.machineTime=InitTime.givenTime=time;

                // set static counters to zero
                crew_counter=0; tides_counter=0;
        }
        else if (strcmp(tokenB, INIT_KEYWD_VES_END)==0) {
                tokenC=strtok(NULL,"\t");
                InitStuff.vessel_data.vessel_end=atoi(tokenC);
                // inserts the vessel number
        }
        else if (strcmp(tokenB, INIT_KEYWD_ARR)==0) {
                tokenC=strtok(NULL,"\t");
                strncpy(InitStuff.move_data.arrival, tokenC,MaxCheckLength);
                // insert inital arrive info
```

```
}
else if (strcmp(tokenB,INIT_KEYWD_DEP)==0) {
        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.move_data.depart, tokenC, MaxCheckLength);
        // insert initial depart info
}
else if (strcmp(tokenB, INIT_KEYWD_ROUTE)==0) {
        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.route_data.route_name, tokenC,
                MaxRouteNameLength);
        // insert initial route info
}
else if (strcmp(tokenB, INIT_KEYWD_WATCH)==0) {
        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.crew_data.watch_name,tokenC,2);
        // insert watch letter
}
else if (strcmp(tokenB, INIT_KEYWD_CREW)==0) {
        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.crew_data.crew_position_name[crew_counter],
                tokenC,crew_name_length);
        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.crew_data.crew_name[crew_counter],
                tokenC,crew_name_length);
        crew_counter++;
        // put in crew position and persons name
}
else if (strcmp(tokenB, INIT_KEYWD_TIDE)==0) {
        char *junk;
        junk=strtok(NULL,"\t"); // junk high
        junk=strtok(NULL,"\t"); // junk height
        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.tides_data.high[tides_counter].height,
                tokenC, tides_length);
        junk=strtok(NULL,"\t"); // junk time
        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.tides_data.high[tides_counter].time,
                tokenC, tides_length);
        junk=strtok(NULL,"\t"); // junk low
        junk=strtok(NULL,"\t"); // junk height

        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.tides_data.low[tides_counter].height,
                tokenC, tides_length);

        junk=strtok(NULL,"\t"); // junk time

        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.tides_data.low[tides_counter].time,
                tokenC, tides_length);
        tides_counter++;
        // read in tides information

}
else if (strcmp(tokenB, INIT_KEYWD_DONE)==0) {
        // now pass this structure off to the DIDP

        DataInitializationDataPacket * DIDP =new
        DataInitializationDataPacket (InitStuff.crew_data,
```

```
                    InitStuff.tides_data,
                    InitStuff.move_data,
                    InitStuff.route_data,
                    InitStuff.vessel_data );
                    assert (DIDP!=NULL);
                    DIDP->initialize_time=InitTime ;
                    //FerryData_global.insert (DIDP);
                    collection_init[which_file-1].insert(DIDP);
                }

                else printf("Unknown entry %s \n", tokenB);


                return 0;
        }
        int HandleMovement (char data_line[MAXDATALINE], int which_file){

                OccurTime move_time;
                char point_name[MAXCOMMANDLENGTH],
point_type[MAXCOMMANDLENGTH];
                char  *token;
                token=strtok(data_line,"\t");  // MOVEMENT
                token=strtok(NULL,"\t"); // point name
                strncpy (point_name, token,MAXCOMMANDLENGTH);
                token=strtok(NULL,"\t"); // point type
                strncpy(point_type, token,MAXCOMMANDLENGTH);
                token=strtok(NULL,"\t"); // machineTime
                move_time.machineTime=atol(token);
                token=strtok(NULL,"\t"); // givenTime
                move_time.givenTime=atol(token);
                MovementDataPacket *MDP = new
                        MovementDataPacket(point_name, point_type,move_time);
                collection_move[which_file-1].insert(MDP);

                return 0;
        }
        int HandleIncident (char data_line[MAXDATALINE], int which_file){

                OccurTime incident_time;
                char incident_name[MAXCOMMANDLENGTH], *token,
                        comment[MaxCommandLength];

                token=strtok(data_line,"\t");  // INCIDENT
                token=strtok(NULL,"\t"); // incident name
                strncpy (incident_name, token,MAXCOMMANDLENGTH);
                token=strtok(NULL,"\t"); // machineTime
                incident_time.machineTime=atol(token);
                token=strtok(NULL,"\t"); // givenTime
                incident_time.givenTime=atol(token);
                token=strtok(NULL,"\t"); // comment
        //      mstrncpy(comment, token,strlen(token)+2);   // problem is here
                IncidentsDataPacket *IDP = new
                        IncidentsDataPacket(incident_name, incident_time,token);
        //              IncidentsDataPacket(incident_name, incident_time,comment);
                /* incident_name gets overwritten by the call to strncpy */
                collection_incid[which_file-1].insert(IDP);
                return 0;

        }
```

```
int HandleDrills (char data_line[MAXDATALINE], int which_file){
        OccurTime drill_time;
        char drill_name[MAXCOMMANDLENGTH],
*token,comment[MaxCommentLength];
        token=strtok(data_line,"\t");  // DRILL
        token=strtok(NULL,"\t"); // drill name
        strncpy (drill_name, token,MAXCOMMANDLENGTH);
        token=strtok(NULL,"\t"); // machineTime
        drill_time.machineTime=atol(token);
        token=strtok(NULL,"\t"); // givenTime
        drill_time.givenTime=atol(token);
        token=strtok(NULL,"\t"); // comment
//      strncpy(comment, token,MaxCommentLength);
        DrillsDataPacket *DDP = new
                DrillsDataPacket(drill_name, drill_time,token);
        collection_drills[which_file-1].insert(DDP);


        return 0;
}

// weather
int HandleWeather (char data_line[MAXDATALINE], int which_file){

        OccurTime weatherTime;
        WeatherInfo weatherData;
        char  *token;


        token=strtok(data_line,"\t");  // WEATHER
        token=strtok(NULL,"\t"); // clouds
        strncpy (weatherData.Clouds, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // precip
        strncpy (weatherData.Precip, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // visability
        strncpy (weatherData.Vis, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // wind direction
        strncpy (weatherData.WindDir, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // wind speed
        strncpy (weatherData.WindSpeed, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // sea
        strncpy (weatherData.Sea, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // temp
        strncpy (weatherData.Temp, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // barometer
        strncpy (weatherData.Barom, token,MaxWeatherLength);

        token=strtok(NULL,"\t"); // machineTime
        weatherTime.machineTime=atol(token);
        token=strtok(NULL,"\t"); // givenTime
        weatherTime.givenTime=atol(token);

        WeatherDataPacket *WDP= new
                WeatherDataPacket (weatherData,weatherTime);
        collection_weath[which_file-1].insert(WDP);

        return 0;

}
```

COMBINE.MAK

```
.AUTODEPEND

#               *Translator Definitions*
CC = bcc +COMBINE.CFG
TASM = TASM
TLIB = tlib
TLINK = tlink
LIBPATH = C:\BORLANDC\LIB;C:\BORLANDC\TVISION\LIB
INCLUDEPATH = C:\BORLANDC\INCLUDE;C:\BORLANDC\TVISION\INCLUDE


#               *Implicit Rules*
.c.obj:
  $(CC) -c {$< }

.cpp.obj:
  $(CC) -c {$< }

#               *List Macros*


EXE_dependencies =  \
 combdat.obj \
 c_funcr.obj \
 combine.obj \
 frrydat2.obj

#               *Explicit Rules*
combine.exe: combine.cfg $(EXE_dependencies)
  $(TLINK) /v/x/c/d/P-/C/L$(LIBPATH) @&&|
c0l.obj+
combdat.obj+
c_funcr.obj+
combine.obj+
frrydat2.obj
combine
                # no map file
tv.lib+
cl.lib
|


#               *Individual File Dependencies*
combdat.obj: combine.cfg combdat.cpp

c_funcr.obj: combine.cfg c_funcr.cpp

combine.obj: combine.cfg combine.cpp

frrydat2.obj: combine.cfg frrydat2.cpp

#               *Compiler Configuration File*
combine.cfg: combine.mak
  copy &&|
-ml
-f-
```

```
-ff-
-W
-vi-
-wpro
-weas
-wpre
-I$(INCLUDEPATH)
-L$(LIBPATH)
-DCOMBINE
-P
| combine.cfg
```

CONSTS.H

```
// This file is consts.h and is used in the ferrylog and combine program

/* this file contains basic numeric and text constants that are used
throughout these programs
*/


#if!defined(__CONSTS_H)
#define __CONSTS_H
// constant declarations



// constant declarations
const int tides_length=15;
const int tides_entries=3;

const int Interior_Stop=1;
const int Interior_OK =0;
const int crew_name_length =50;
const int crew_positions =11;
const int MaxCheckPoints=10;
const int MaxCommandLength=15;
        /* total possible number of displayed checkpoints */
const int MaxCommentLength=250;
const int MaxWeatherLength=50;
const int MaxRouteData_ad=15;
const int MaxInciNameLength=85; // max length an incident name can take
const int MaxDrillNameLength=30; // max length a drill name can take
/* total possible number of arrival and departure points stored for the
current route */

const int MaxRouteData_ck=40;
/* total possible number of check points stored for the
current route */

const int MaxRouteNameLength=80;  // max length of route name

const int MaxCheckLength=10;
const int MaxDrillLength=15;
const int ArrivalIndicator =(-1);
const int DepartIndicator=(-2);
const int MaxVesselNameLength=15;


const int VALID=1;
const int INVALID=0;
const int DISK_VALID=0;
const int DISK_INVALID=(-1);
const int No=0;
const int Yes=1;
const int On=1;
const int Off=0;
const int sound_record=0;
const int sound_oops=1;
const int sound_alert=2;
```

```
// command declarations

// these constants with a msg prefix indicates those commands sent
// out by the message command.

const int msgTidesViewer =1500;
const int msgCrewViewer=1501;
const int msgMovementViewer=1502;
const int msgDrillsViewer=1503;
const int msgIncidentsViewer=1504;
const int msgFireInitup=1505;
const int msgWeatherEnter=1506;
const int msgPrinterOptionsBox=1507;
const int msgLogViewer=1508;


// these with the cm prefix indicate that the command did not
// originate from the message command
const int cmCloseDesktop=1300;  // command to close up the inital desktop
const int cmTidesViewer =1301; // command from menu for tides viewer
const int cmCrewViewer =1302;
const int cmMovementViewer=1303;
const int cmArrival=1304;
const int cmDeparture=1305;
const int cmjunk=1306;
const int cmOccurBefore=1307;
const int cmOccurNow=1308;
const int cmIncidentsViewer=1309;
const int cmDrillsViewer=1310;
const int cmdrillAbS=1311;
const int cmdrillRB=1312;
const int cmdrillF=1313;
const int cmincidentSLIP=1314;
const int cmincidentVESSEL=1315;
const int cmincidentAUTO=1316;
const int cmincidentCREW=1317;
const int cmincidentPASS=1318;
const int cmincidentWEATHER=1319;
const int cmincidentMECHAN=1320;
const int cmincidentPOLICE=1321;
const int cmincidentCG=1322;
const int cmincidentOTHER=1323;
const int cmWeatherEnter=1324;
const int cmWeatherRecord=1325;
const int cmWeatherJunk=1326;
const int cmPrinterOptionsBox=1327;
const int cmUPTIME=1328;  // INCREMENTS TIME COUNTER
const int cmDOWNTIME=1329; // DECREMENTS TIME COUNTER
const int cmEndProgram=1330;  // sets up end command sequence
const int cmLogViewer=1331;
const int cmAboutView=1332;
const int ButtonPopBottom=2000;                    // commands
const int ButtonPopTop=ButtonPopBottom+MaxCheckPoints;  // commands

const int cmArrivalChooseBt =2200;
const int cmDepartureChooseBt =2400;

/***** these are integers to identify the vessel  ****/
// if any more vessels are added, the should be
```

```
// added after yakima and in numeric sequence
// and the constant titled const int id_ves_end
// should be modified appropriately

const int id_ves_start=3000;
const int id_ves_cath= 3000;
const int id_ves_chel= 3001;
const int id_ves_elwh= 3002;
const int id_ves_eveg= 3003;
const int id_ves_hiyu= 3004;
const int id_ves_hyak= 3005;
const int id_ves_illa= 3006;
const int id_ves_issa= 3007;
const int id_ves_kala= 3008;
const int id_ves_kale= 3009;
const int id_ves_kits= 3010;
const int id_ves_kitt= 3011;
const int id_ves_klah= 3012;
const int id_ves_klic= 3013;
const int id_ves_nisq= 3014;
const int id_ves_olym= 3015;
const int id_ves_quin= 3016;
const int id_ves_rhod= 3017;
const int id_ves_seal= 3018;
const int id_ves_skag= 3019;
const int id_ves_spok= 3020;
const int id_ves_till= 3021;
const int id_ves_tyee= 3022;
const int id_ves_wall= 3023;
const int id_ves_yaki= 3024;
const int id_ves_end=3024;  // highest numbered command to id the vessels


// if any more routes are added, the should be
// added after "id_route_poit" and in numeric sequence
// and the constant titled const int id_route_end
// should be modified appropriately
// consts to identify the route
const int id_route_start=3100; // lowest numbered command to id the routes
const int id_route_anac=3100;
const int id_route_town=3101;
const int id_route_muki=3102;
const int id_route_edmd=3103;
const int id_route_seaw=3104;
const int id_route_seab=3105;
const int id_route_fvso=3106;
const int id_route_seav=3107;
const int id_route_poit=3108;
const int id_route_end=3108;  // highest numbered command to id the routes


// vessel end constants

const int id_ves_end_only=3200;
const int id_ves_end_one=3201;
const int id_ves_end_two=3202;

// watch consts
```

```
const int id_watch_A=3300;
const int id_watch_B=3301;
const int id_watch_C=3302;
const int id_watch_D=3303;
const int id_watch_E=3304;
const int id_watch_F=3305;
const int id_watch_G=3306;
const int id_watch_H=3307;
const int id_watch_I=3308;
const int id_watch_J=3309;
const int id_watch_K=3310;


/****  text constants ****/


// drills
const char ABANDON_TXT[]="ABANDON_DRILL";
const char RESCUE_TXT[]="RESCUE_BOAT_DRILL";
const char FIRE_TXT[]="FIRE_DRILL";

// incidents

const char INCI_SLIP[]="WAIT_FOR_SLIP";
const char INCI_VESSEL[]="HEAVY_VESSEL_TRAFFIC";
const char INCI_AUTO[]="HEAVY_AUTO_TRAFFIC";
const char INCI_CREW[]="CREWING_PROBLEM";
const char INCI_PASS[]="PASSENGER_PROBLEM";
const char INCI_WEATHER[]="WEATHER,TIDES,SEAS";
const char INCI_MECHAN[]="MECHANICAL_PROBLEMS";
const char INCI_POLICE[]="POLICE,AMBULANCE,ETC";
const char INCI_COASTGU[]="COAST_GUARD_REQUEST";
const char INCI_OTHER[]="OTHER";

// vessel Names

const char VES_NAME_CATH[]="Cathlamet";
const char VES_NAME_CHEL[]="Chelan";
const char VES_NAME_ELWH[]="Elwha";
const char VES_NAME_EVEG[]="Ev. Green";
const char VES_NAME_HIYU[]="Hiyu";
const char VES_NAME_HYAK[]="Hyak";
const char VES_NAME_ILLA[]="Illahee";
const char VES_NAME_ISSA[]="Issaquah";
const char VES_NAME_KALA[]="Kalama";
const char VES_NAME_KALE[]="Kaleetan";
const char VES_NAME_KITS[]="Kitsap";
const char VES_NAME_KITT[]="Kittitas";
const char VES_NAME_KLAH[]="Klahowya";
const char VES_NAME_KLIC[]="Klickitat";
const char VES_NAME_NISQ[]="Nisqually";
const char VES_NAME_OLYM[]="Olympic";
const char VES_NAME_QUIN[]="Quinault";
const char VES_NAME_RHOD[]="Rhodie";
const char VES_NAME_SEAL[]="Sealth";
const char VES_NAME_SKAG[]="Skagit";
const char VES_NAME_SPOK[]="Spokane";
const char VES_NAME_TILL[]="Tillikum";
```

```
const char VES_NAME_TYEE[]="Tyee";
const char VES_NAME_WALL[]="Walla Walla";
const char VES_NAME_YAKI[]="Yakima";

//route names
const char ROUTE_NAME_ANAC[]="Anacortes, San Juans, Sidney, Friday Harbor";
const char ROUTE_NAME_TOWN[]="Port Townsend Keystone";
const char ROUTE_NAME_MUKI[]="Mukilteo Clinton";
const char ROUTE_NAME_EDMD[]="Edmonds Kingston";
const char ROUTE_NAME_SEAW[]="Seattle Bainbridge Island (Winslow)";
const char ROUTE_NAME_SEAB[]="Seattle Bremerton";
const char ROUTE_NAME_FVSO[]="Fauntleroy Vashon Southworth";
const char ROUTE_NAME_SEAV[]="Seattle Vashon";
const char ROUTE_NAME_POIT[]="Point Defiance Tahlequah";

// CREW position names
const char CREW_POS_0_NAME[]="MASTER";
const char CREW_POS_1_NAME[]="CHIEF MATE";
const char CREW_POS_2_NAME[]="SECOND MATE";
const char CREW_POS_3_NAME[]="QUARTERMASTER";
const char CREW_POS_4_NAME[]="BOSUN";
const char CREW_POS_5_NAME[]="ABLE BODY 1";
const char CREW_POS_6_NAME[]="ABLE BODY 2";
const char CREW_POS_7_NAME[]="ORDINARY 1";
const char CREW_POS_8_NAME[]="ORDINARY 2";
const char CREW_POS_9_NAME[]="ORDINARY 3";
const char CREW_POS_10_NAME[]="ORDINARY 4";



// WATCH names
const char ID_WATCH_A[]="A";
const char ID_WATCH_B[]="B";
const char ID_WATCH_C[]="C";
const char ID_WATCH_D[]="D";
const char ID_WATCH_E[]="E";
const char ID_WATCH_F[]="F";
const char ID_WATCH_G[]="G";
const char ID_WATCH_H[]="H";
const char ID_WATCH_I[]="I";
const char ID_WATCH_J[]="J";
const char ID_WATCH_K[]="K";

// these text constants identify the data labels

const char ACTION_MOVEMENT[]="MOVEMENT\0";
const char ACTION_INCIDENTS[]="INCIDENT\0";
const char ACTION_DRILLS[]="DRILL\0";
const char ACTION_ARRIVE[]="ARRIVE\0";
const char ACTION_DEPART[]="DEPART\0";
const char ACTION_CHKPOINT[]="CHECKPOINT\0";
const char ACTION_INITIALIZE[]="INITIALIZE\0";
const char ACTION_WEATHER[]="WEATHER\0";

// keywords for the initialization data as it is written to disk

const char INIT_KEYWD_VES_NAME[]="VESSEL NAME";
const char INIT_KEYWD_VES_END[]="VESSEL END";
const char INIT_KEYWD_ROUTE[]="ROUTE";
```

```
const char INIT_KEYWD_ARR[]="INITIAL ARRIVAL";
const char INIT_KEYWD_DEP[]="INITIAL DEPARTURE";
const char INIT_KEYWD_WATCH[]="WATCH";
const char INIT_KEYWD_CREW[]="CREW";
const char INIT_KEYWD_TIDE[]="TIDES";
const char INIT_KEYWD_TIDEh[]="HIGH";
const char INIT_KEYWD_TIHI[]="Height";
const char INIT_KEYWD_TIDEl[]="LOW";
const char INIT_KEYWD_TITM[]="Time";
const char INIT_KEYWD_DONE[]="DONE";

// more character constants
const char FILE_MARKER_TOP[]="FILE TOP";
const char FILE_MARKER_BOTTOM[]="FILE BOTTOM";
#endif  //__CONSTS_H
```

```
/*-------------------------------------------------*/
/*                                                 */
/*      Turbo Vision 1.0                           */
/*      Turbo Vision FileViewer Demo Support File  */
/*      Copyright (c) 1991 by Borland International */
/*                                                 */
/*-------------------------------------------------*/

/* this source file was modified from the TurboVision FileView Demo
        by Erik H. Beck
        December, 1992                  */
// this file is used by the ferrylog program

/* this file is datview.cpp. The source code contained herein enables the
Ferrylog user to view data recorded during the session. Because of
the way the collections are currently used, there is some delay from the
time that data is entered to the time that it shows up on this screen. This
File Viewer views the file Visual.dat that is written by every session of the
Ferrylog program. Therefore, at the end of a session, the visual.dat file
should be erased. The collections in the file ferrydat2.cpp record the
data into the visual.dat file.

*/

#define Uses_MsgBox
#define Uses_TKeys
#define Uses_TScroller
#define Uses_TDrawBuffer
#define Uses_TRect
#define Uses_TProgram
#define Uses_TDeskTop

#include <tv.h>
//__link(RScroller)
//__link(RScrollBar)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include <fstream.h>

#include "consts.h"
#include "datview.h"
// global fstream

extern fstream VisFile;  // read and append modes


LogDataViewer::LogDataViewer( const TRect& bounds,
                                                TScrollBar *aHScrollBar,
                                                TScrollBar *aVScrollBar) :
        TScroller( bounds, aHScrollBar, aVScrollBar )
{
        growMode = gfGrowHiX | gfGrowHiY;
```

```
        isValid = True;
        readFile();

}

LogDataViewer::~LogDataViewer()
{
        destroy (fileLines);
}

void LogDataViewer::draw()
{

        char *p;

        ushort c =      getColor(0x0301);
        for( int i = 0; i < size.y; i++ )
                {
                TDrawBuffer b;
        b.moveChar( 0, ' ', c, size.x );

                if( delta.y + i < fileLines->getCount() )
                        {
                        char s[MaxLineLength+1];
                        p = (char *)( fileLines->at(delta.y+i) );
                        if( p == 0 || strlen(p) < delta.x )
                                s[0] = EOS;
                        else
                {
                strncpy( s, p+delta.x, size.x );
                if( strlen( p + delta.x ) > size.x )
                                        s[size.x] = EOS;
                                }
                        b.moveStr( 0, s, c );
                        }
                writeBuf( 0, i, size.x, 1, b );
                }
}

void LogDataViewer::scrollDraw()
{
        TScroller::scrollDraw();
        draw();
}

void LogDataViewer::readFile(void)
{

        limit.x = 0;
        fileLines = new TLineCollection(5, 5);

        if( !VisFile )
                {
                messageBox( "Invalid drive or directory", mfError | mfOKButton );
                isValid = False;
                }
        else
                {
                VisFile.seekg(0,ios::beg); // go to file beginning
```

```
                char line[MaxLineLength+1];
                while( !lowMemory() &&
                        !VisFile.eof() &&
                        VisFile.get( line, sizeof line ) != 0
                      )
                      {
                      char c;
                      VisFile.get(c);           // grab trailing newline
                      limit.x = max( limit.x, strlen( line ) );
                      fileLines->insert( newStr( line ) );
                      }
                isValid = True;
                }
        limit.y = fileLines->getCount();
        VisFile.clear(); // clears state of EOF
}

void LogDataViewer::setState( ushort aState, Boolean enable )
{
        TScroller::setState( aState, enable );
        if( enable && (aState & sfExposed) )
                setLimit( limit.x, limit.y );
}

Boolean LogDataViewer::valid( ushort )
{
        return isValid;
}



LogDataWindow::LogDataWindow(short winNum) :
        TWindow( TProgram::deskTop->getExtent(), "Log View",winNum),
        TWindowInit( &LogDataWindow::initFrame )
{
        options |= ofTileable;
        TRect r( getExtent() );
        r.grow(-1, -1);
        insert(new LogDataViewer( r,
        standardScrollBar(sbHorizontal | sbHandleKeyboard),
        standardScrollBar(sbVertical | sbHandleKeyboard) ) );
}
```

DATVIEW.H

```
/*-----------------------------------------*/
/*                                         */
/*  Turbo Vision 1.0                       */
/*  Copyright (c) 1991 by Borland International        */
/*                                         */
/*  Fileview.h:  Header file for fileview.cpp.         */
/*-----------------------------------------*/

// modified to datview.h by erik beck
// this header file is used by the ferrylog program
/* this file gives class declarations for the data viewer view */


#if !defined( __DATVIEW_H )
#define __DATVIEW_H

#define Uses_TNSCollection
#define Uses_TScroller
#define Uses_TWindow
#include <tv.h>



class TLineCollection : public TNSCollection
{

public:

   TLineCollection(short lim, short delta) : TNSCollection(lim, delta) {}
   virtual void  freeItem(void *p) { delete p; }

private:

   //virtual void *readItem( ipstream& ) { return 0; }
   //virtual void writeItem( void *, opstream& ) {}

};

class LogDataViewer : public TScroller
{

public:

   TNSCollection *fileLines;
   Boolean isValid;
   LogDataViewer( const TRect& bounds,
         TScrollBar *aHScrollBar,
         TScrollBar *aVScrollBar
       );
   ~LogDataViewer();
   LogDataViewer( StreamableInit ) : TScroller(streamableInit) { };
   void draw();
   void readFile(void);
   void setState( ushort aState, Boolean enable );
   void scrollDraw();
   Boolean valid( ushort command );
```

```
private:

 // virtual const char *streamableName() const
//     { return name; }

protected:

//   virtual void write( opstream& );
 //  virtual void *read( ipstream& );

public:

 //  static const char * const name;
 //  static TStreamable *build();

};

class LogDataWindow : public TWindow
{

public:

   LogDataWindow(short);

};

const int MaxLineLength = 256;

#endif
```

DRIVER3.CPP

```
// "main" file for the FerryLog program

// this source file, driver3.cpp, contains the main module and two
// TApplication modules that are used to drive the interface.
// In a sense, this file 'drives' all the other source files to do
// their jobs.

#define Uses_TApplication
#define Uses_TStatusLine
#define Uses_TMenuBar
#define Uses_TDeskTop
#define Uses_TStatusDef
#define Uses_TStatusItem
#define Uses_TSubMenu
#define Uses_TMenuItem
#define Uses_TKeys
#define Uses_TProgram
#define Uses_TObject
#define Uses_MsgBox
#define Uses_TLabel
#define Uses_TButton
#include "c_func2.h"
#include "miscdat.h"
#include "guiwx.h"
#include "misgui1.h"
#include "datview.h"
#include "frrygui3.h"
#include "pstream.h"
#include <tv.h>


#include <sys\stat.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include "gadgets.h"
//#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <dos.h>

static short winNumber=0;
int RepeatFlag;

// test global

//MoveInfo locations;
extern  int _Cdecl directvideo=0;
extern unsigned _stklen=10240U;  // increase size of heap to 10K

prnstream PrinterStream(0);
PrintOptionData POD;
fstream VisFile ("Visual.dat",ios::in|ios::app); //read and append modes
```

```
FerryFileNames DataFileNames (void); // function declaration
int DataFileCheck (FerryFileNames origname, char filetype, int rep_number);

// need to execute these things so that they have global definition

FerryFileNames outfiles=DataFileNames(); // execute function
int final_file_check=DataFileCheck(outfiles,'F',0); // execute
int temp_file_check=DataFileCheck(outfiles,'T',0); // execute
FerryDataCollection  FerryData_global (5,2);
int FinalReportPrint(char *);

FlopInfo FlopCheck (char * FileName); // check floppy status declaration

/******    declaration of the driver class for main interface  *************/
class DriverClassM: public TApplication {
public:
        DriverClassM(DataStorageClass &);
        DataStorageClass ferry_data;
        static TStatusLine * initStatusLine (TRect r);
        static TMenuBar *initMenuBar (TRect r);
        virtual void handleEvent (TEvent &event);
        void myTidesWindow();
        void myCrewWindow();
        void myMoveWindow();
        void myDrillsWindow();
        void myIncidentsWindow();
        void myWeatherReport();
        void myPrinterOptions();
        virtual void idle();
        void aboutDlgBox(void);
        virtual void outOfMemory(void);
        void myshutdown(void);
        int CopyFloppy(void);
        void LogDataViewer(void);
        void FloppyProblem(char *FlopTxt);
        ushort DisplayPrinterError (const char *errorMsg);
private:
        TClockView *clock;
        THeapView *heap;
};

/******  end of declaration  ********/

/***  Driver class I declaration   Initializes the program info  ****/


class DriverClassI: public TApplication {
public:
        DriverClassI(DataStorageClass *);
        static TStatusLine * initStatusLine (TRect r);
        static TMenuBar *initMenuBar (TRect r);
        virtual void handleEvent (TEvent &event);
        DataStorageClass *ferry_setup;
        virtual void idle(void);
        virtual void outOfMemory(void);
        void InitRepeat (void);
        void aboutDlgBox(void);
```

```
        private:
                TClockView *clock;
        };

        /****    end of class declarations  *****/

        /******    definition section   *********/

        /* test driver section of the code for main section */
        DriverClassM::DriverClassM(DataStorageClass &ferry_pass)
:TProgInit(DriverClassM::initStatusLine,
                        DriverClassM::initMenuBar,
                        DriverClassM::initDeskTop) {

                TRect r = getExtent();              // Create the clock view.
                                                    // put on status line
                ferry_data=ferry_pass;
                r.a.x = r.b.x - 29;
                r.a.y = r.b.y - 1;
                showMarkers=True; // modified for monochrome
                clock = new TClockView( r );
                insert(validView(clock));
                // insert heap view for testing
                r= getExtent();
                r.a.x = r.b.x - 19;
                r.b.y = r.a.y + 1;
                heap= new THeapView (r);
                insert (validView(heap));

        }

        void DriverClassM::idle(void) {
                TProgram::idle();
                clock->update();
                heap->update();
        }
        void DriverClassM::outOfMemory (void) {
                messageBox("Not enough memory to complete operation.", mfError |
                        mfOKButton);
        }
        void DriverClassM::FloppyProblem(char *FlopTxt) {
                messageBox((const char *)FlopTxt,mfError|mfOKButton);
        }



        TStatusLine *DriverClassM::initStatusLine (TRect r)
        {
                r.a.y=r.b.y -1;
                return new TStatusLine (r,
                *new TStatusDef(0,0xffff) +
                *new TStatusItem (0,kbF10, cmMenu) +
                *new TStatusItem ("~Alt-X~ Exit", kbAltX, cmEndProgram) +
                *new TStatusItem ("~Alt-F3~ Close", kbAltF3, cmClose) );
        }
        TMenuBar *DriverClassM::initMenuBar(TRect r) {
                r.b.y=r.a.y +1;
                return new TMenuBar (r,
```

```
        *new TSubMenu("~A~bout", kbAltA) +
        *new TMenuItem("Program",cmAboutView,kbAltP, hcNoContext,"Alt P") +
        *new TSubMenu ("~D~isplay", kbAltD) +
        *new TMenuItem ("Tides", cmTidesViewer, kbF1,
                hcNoContext, "F1") +
        newLine() +
        *new TMenuItem ("Crew", cmCrewViewer, kbF2, hcNoContext,"F2") +
        *new TSubMenu ("~M~di", kbAltM) +
        *new TMenuItem ("Movement", cmMovementViewer, kbF3, hcNoContext,
                "F3") +
        newLine() +
        *new TMenuItem("Incidents", cmIncidentsViewer, kbCtrlF3, hcNoContext,
                "Ctrl-F3") +
        newLine() +
        *new TMenuItem("Drills", cmDrillsViewer, kbShiftF3, hcNoContext,
                "Shift-F3") +
        *new TSubMenu("~W~x",kbAltW) +
        *new TMenuItem("Weather",cmWeatherEnter,kbF4,hcNoContext,
                "F4")+
        *new TSubMenu("~O~ptions",kbAltO) +
        *new TMenuItem("Printer",cmPrinterOptionsBox,kbF5,hcNoContext,"F5")+
        *new TSubMenu("m~I~sc", kbAltI) +
        *new TMenuItem("Log View",cmLogViewer, kbF6,hcNoContext,"F6")

        );

}
int DriverClassM::CopyFloppy(void) {
        // copy info onto the floppy
        int returnValue=VALID;

        FlopInfo floppyInfo;
        floppyInfo=FlopCheck(outfiles.FinalFileName);
        if (floppyInfo.FlopStatusInt==DISK_VALID) {

                char floppyname [20];
                strncpy(floppyname, " ",20);
                strcpy(floppyname,"a:\\");
                strcat(floppyname,outfiles.FinalFileName);

                int cp_call=mycopy(outfiles.TempFileName, floppyname);
/*
                if (cp_call !=0 ) {

                        fprintf(test_file,
                                "floppy copy returned error. Value is: %d \n",
                                cp_call);
                }
*/
        }
        else {
                // pop a window
                FloppyProblem(floppyInfo.FlopStatusTxt);
                // change return value to invalid
                returnValue=INVALID;

        }

        return returnValue;
```

```
}
void DriverClassM::handleEvent (TEvent &event) {

        TApplication::handleEvent(event); //act like a base

        // add shut down procedure
        if (event.what==evCommand) {
                switch (event.message.command) {
                        case cmLogViewer:

                                clearEvent(event);
                                LogDataWindow *exist_LDW=
                                (LogDataWindow *)message
                                (deskTop, evBroadcast,
                                msgLogViewer,0);

                                if (exist_LDW==0)
                                        LogDataViewer();
                                else exist_LDW->select();

                        break;
                        case cmEndProgram:
                                clearEvent(event);
                                myshutdown();
                                // execute some commands, then quit
                                // quit command gets put in later in the
                                // myshutdown() routine
                        break;

                        case cmTidesViewer:
                                clearEvent(event);
                                TidesViewer *exist_tides_viewer=
                                        (TidesViewer *)message
                                        (deskTop, evBroadcast,
                                        msgTidesViewer,0);

                                if (exist_tides_viewer==0)
                                        myTidesWindow();
                                else exist_tides_viewer->select();
                        break;
                        case cmCrewViewer:
                                clearEvent(event);
                                CrewViewer *exist_crew_viewer=
                                        (CrewViewer*)message
                                        (deskTop, evBroadcast,
                                        msgCrewViewer,0);
                                if (exist_crew_viewer==0)
                                        myCrewWindow();
                                else exist_crew_viewer->select();
                        break;
                        case cmMovementViewer:
                                clearEvent(event);
                                MovementViewer *exist_move_viewer=
                                        (MovementViewer *)message
                                        (deskTop, evBroadcast,
                                        msgMovementViewer, 0);
                                if (exist_move_viewer==0)
                                        myMoveWindow();
```

```
                    else exist_move_viewer-> select();
                break;
                case cmDrillsViewer:
                    clearEvent(event);
                    DrillsViewer *exist_drills_viewer=
                    (DrillsViewer *)message
                    (deskTop, evBroadcast,
                    msgDrillsViewer, 0);
                    if(exist_drills_viewer==0)
                        myDrillsWindow();
                    else exist_drills_viewer->select();
                break;

                case cmIncidentsViewer:
                    clearEvent(event);
                    IncidentsViewer *exist_incidents_viewer=
                    (IncidentsViewer *)message
                    (deskTop, evBroadcast,
                    msgIncidentsViewer, 0);
                    if(exist_incidents_viewer==0)
                        myIncidentsWindow();
                    else exist_incidents_viewer->select();
                break;
                case cmAboutView:
                    clearEvent(event);
                    aboutDlgBox();
                break;
                case cmWeatherEnter:
                    clearEvent(event);
                    WeatherReport *exist_weather_report=
                        (WeatherReport *)message
                        (deskTop, evBroadcast,
                        msgWeatherEnter,0);

                    if (exist_weather_report==0)
                        myWeatherReport();
                    else exist_weather_report->select();
                break;
                case cmPrinterOptionsBox:
                    clearEvent(event);
                    PrintOptions *exist_print_op=
                        (PrintOptions *)message
                        (deskTop, evBroadcast,
                        msgPrinterOptionsBox,0);

                    if (exist_print_op==0)
                        myPrinterOptions();
                    else exist_print_op->select();
                break;

                default:
                    return;

            }
            clearEvent(event);
        }
}
void DriverClassM::LogDataViewer(void) {
        // routine to display current log information
```

```
        // set initial size and position
        TRect WinSize( 0, 0, 65, 13 );

        LogDataWindow *VWindow =
                new LogDataWindow (++winNumber);

        // put window into desktop and draw it
        deskTop->insert(validView(VWindow));

}

void DriverClassM::myshutdown(void) {
        // do a few things, then shut down class

        FerryData_global.writeAll();
        FerryData_global.fileclose();
        // try printing final report here
        if (POD.finalData==On) {
                int PrintReturnValue;
                PrintReturnValue=
                        FinalReportPrint(outfiles.TempFileName);
        }



        // copy and delete the temporary file using dos commands


        int cp_call=mycopy(outfiles.TempFileName, outfiles.FinalFileName);
        // dos copy

/*      if (cp_call !=0 ) {
                fprintf(test_file,
                        "Disk final copy returned error.  Value is: %d \n",
                        cp_call);
        }
*/
        //now copy to a floppy
        int rvalue=CopyFloppy();
        int attempts=0;
        while(rvalue!=VALID) {
                rvalue=CopyFloppy();
        }

        // Now check the copied file before deleting the copy (final)
        int temp_ret_status, final_ret_status;
        struct stat temp_stat, final_stat;
        temp_ret_status=stat(outfiles.TempFileName, &temp_stat);
        final_ret_status=stat(outfiles.FinalFileName, &final_stat);

        if (temp_ret_status==0 && final_ret_status==0) {
                if (temp_stat.st_size==final_stat.st_size) {
                        // ok files are same size, so should have copied
                        // now delete temp file
                        char cp_command[22];
                        strncpy(cp_command, "del ",22);
                        strcat(cp_command, outfiles.TempFileName);
                        system(cp_command);
```

```
                    // still hangs the compaq ???

                }
        }


        // put quit command back on the event queue
        TEvent event;
        event.what=evCommand;
        event.message.command=cmQuit;
        putEvent(event);



}


void DriverClassM::myTidesWindow()
{
        TidesInfo tideit;
        TRect r(0,0,20,21);

        TidesViewer *tideswindow =new TidesViewer (r, ++winNumber,
                ferry_data.tides_data);
        assert (tideswindow!=0);
        deskTop->insert(validView(tideswindow));

}
void DriverClassM::myWeatherReport(void)
{
        WeatherReport *weather =new WeatherReport();
        assert (weather!=0);
        deskTop->insert(validView(weather));

}
void DriverClassM::myPrinterOptions(void) {

        ushort rv; // return value from error window
        PrintOptions *PO =new PrintOptions();
        assert(PO!=0);
        //PrintOptionData POD;
        PO->setData(&POD);
        ushort control =deskTop->execView(validView(PO));
        if (control!=cmCancel)
                PO->getData(&POD);
        destroy (PO);

        // now act on information in POD;
        // test block

        if ( POD.spewData==1){
                ushort buttonPush;
                PrinterStatusInfo PStat=PrinterStatus ();
                char temp[20];
                strncpy (temp, " ", 20);
                strcpy (temp, PStat.PrinterStatusText);
                if (PStat.PrinterStatusInt!=Yes) {
                        buttonPush=
                                DisplayPrinterError( (const char *) temp);
```

```
                        if (buttonPush==cmCancel) {
                                SpewPrintOff();
                                SummaryPrintOff();
                        }
                }
                else SpewPrintOn();


        }
        else if (POD.spewData==0) {
                SpewPrintOff();
        }

        if ( POD.finalData==1){
                ushort buttonPush;
                PrinterStatusInfo PStat=PrinterStatus ();
                char temp[20];
                strncpy (temp, " ", 20);
                strcpy (temp, PStat.PrinterStatusText);
                if (PStat.PrinterStatusInt!=Yes) {
                        buttonPush=
                                DisplayPrinterError( (const char *) temp);
                        if (buttonPush==cmCancel) {
                                SpewPrintOff();
                                SummaryPrintOff();
                        }
                }
                else SummaryPrintOn();


        }
        else if (POD.spewData==0) {
                SummaryPrintOff();
        }


}
ushort DriverClassM::DisplayPrinterError (const char *errorMsg){
        ushort control=0;
        PrinterError *prErr=new PrinterError(errorMsg);
        assert(prErr!=0);
        //ushort control=(deskTop->execView(validView(prErr)));
        // have to use this because function is static
        control=deskTop->execView (validView(prErr));
        destroy (prErr);
        return control;
}

void DriverClassM::myCrewWindow()
{
        TRect r(2,2,17,11);

        CrewViewer *crewwindow =new CrewViewer (r, ++winNumber,
                ferry_data.crew_data);
        deskTop->insert(validView(crewwindow));
}
void DriverClassM::myMoveWindow() {
        MovementViewer *moveit;
        moveit= new MovementViewer(ferry_data.move_data);
```

```
        deskTop->insert(validView(moveit));
}
void DriverClassM::myDrillsWindow() {
        DrillsViewer *drills;
        drills= new  DrillsViewer();
        deskTop -> insert(validView(drills));
}
void DriverClassM::myIncidentsWindow() {
        IncidentsViewer *incidents;
        incidents=  new  IncidentsViewer() ;
        deskTop -> insert(validView(incidents));

}

/*** test driver definition for the initialization of the data ****/

DriverClassI::DriverClassI(DataStorageClass *ferry_setup_pass) :
        TProgInit(DriverClassI::initStatusLine,
        DriverClassI::initMenuBar,
        DriverClassI::initDeskTop) {

        //add markers around focused controls
        showMarkers=True;

        ferry_setup=ferry_setup_pass;
        TRect r = getExtent();              // Create the clock view.
                                                    // put on status line
        r.a.x = r.b.x - 29;
        r.a.y = r.b.y - 1;


        clock = new TClockView( r );
        insert(validView(clock));

        aboutDlgBox();

        VesselInit *vessels;

        vessels = new VesselInit (&(ferry_setup->vessel_data));
        deskTop -> execView (validView(vessels));
        destroy(vessels);

        VesselEndInit *vesEnd;
        vesEnd = new VesselEndInit ( &(ferry_setup->vessel_data));
        deskTop -> execView (validView(vesEnd));
        destroy (vesEnd);

        RouteInit *routes;
        routes=new RouteInit((&ferry_setup->route_data));
        deskTop ->execView (validView(routes));
        destroy(routes);

        ferry_setup->route_setup();

        DepartureInit *where_depart;
        where_depart=new DepartureInit ( (&ferry_setup->move_data),
                (&ferry_setup->route_data));
        deskTop->execView(validView(where_depart));
        destroy (where_depart);
```

```
        ArrivalInit *where_arrive;
        where_arrive=new ArrivalInit ( (&ferry_setup->move_data),
                (&ferry_setup->route_data));
        deskTop->execView(validView(where_arrive));
        destroy (where_arrive);
        ferry_setup->move_setup();

        WatchInit *watchStuff;
        watchStuff = new WatchInit ( (&ferry_setup->crew_data));
        deskTop->execView (validView (watchStuff));
        destroy(watchStuff);

        CrewInit *crewStuff;
        crewStuff= new CrewInit
                ((&ferry_setup->crew_data));
        deskTop->execView (validView(crewStuff));
        destroy(crewStuff);

        TidesInit *tidesStuff;
        tidesStuff = new TidesInit ((&ferry_setup->tides_data));
        deskTop->execView (validView(tidesStuff));
        destroy(tidesStuff);

        InitRepeat (); // check to see if initialization needs to be
                        // repeated


}
void DriverClassI::InitRepeat(void) {
        RepeatFlag=0; // file scope global integer

        PrimaryDialog *RepeatDialog=new PrimaryDialog(TRect(0,0,75,20),
                "Repeat Request");
        RepeatDialog->insert (new TLabel(TRect(2,2,65,3),
                "Did you make a mistake setting up the watch info?",0));

        TButton *YesButton=new TButton(TRect(2,6,15,8),"Yes",cmYes,bfNormal);
        RepeatDialog->insert(YesButton);
        TButton *NoButton=new TButton(TRect(18,6,33,8),"No",cmNo,bfNormal);
        RepeatDialog->insert(NoButton);
        NoButton->select();
        ushort RepeatStatus= deskTop->execView(validView(RepeatDialog));
        destroy (RepeatDialog);
        if (RepeatStatus==cmYes) RepeatFlag=Yes;
        else RepeatFlag=No;
}
void DriverClassI::aboutDlgBox(void)
{
   TDialog *aboutBox = new TDialog(TRect(0, 0, 39, 13), "About");

   aboutBox->insert(
    new TStaticText(TRect(9, 2, 30, 9),
        "\003Computer Aided Vessel Logbook\n\003\n"     // These strings will be
        "\003Version 1.0\n\003\n"          // concatenated by the compiler.
        "\003Completed December 31, 1992\n\003\n"    // The Ctrl-C centers the line.
        "\003Erik H. Beck"
        )
   );
```

```
  aboutBox->insert(
   new TButton(TRect(14, 10, 25, 12), " OK", cmOK, bfDefault)
  );

  aboutBox->options |= ofCentered;

  deskTop->insert(aboutBox);
  delay(1000);
  aboutBox->close();

}
void DriverClassM::aboutDlgBox(void)
{
  TDialog *aboutBox = new TDialog(TRect(0, 0, 39, 13), "About");

  aboutBox->insert(
   new TStaticText(TRect(9, 2, 30, 9),
        "\003Computer Aided Vessel Log\n\003\n"    // These strings will be
        "\003Version 1.0\n\003\n"          // concatenated by the compiler.
        "\003Completed December 31, 1992\n\003\n"   // The Ctrl-C centers the line.
        "\003Erik H. Beck"
        )
  );

  aboutBox->insert(
   new TButton(TRect(14, 10, 25, 12), " OK", cmOK, bfDefault)
  );

  aboutBox->options |= ofCentered;

  deskTop->execView(aboutBox);

  destroy( aboutBox );
}


void DriverClassI::idle() {
        TProgram::idle();
        clock->update();
}
void DriverClassI::outOfMemory (void) {
        messageBox("Not enough memory to complete operation.", mfError |
                mfOKButton);
}

TStatusLine *DriverClassI::initStatusLine (TRect r)
{
        r.a.y=r.b.y -1;
        return new TStatusLine (r,
        *new TStatusDef(0,0xffff) +
        *new TStatusItem (0,kbF10, cmMenu) );

}
TMenuBar *DriverClassI::initMenuBar(TRect r) {

        r.b.y=r.a.y+1;
        return new TMenuBar(r, NULL);
}
```

```
void DriverClassI::handleEvent (TEvent &event) {
        TApplication::handleEvent(event); //act like a base
        if (event.what==evCommand) {
                if (event.message.command==cmCloseDesktop) {
                        clearEvent (event);
                        event.what=evCommand;
                        event.message.command=cmCancel;
                        putEvent(event);  // close the desktop
                }
        }

}


/*****  main program for driver  *********/
int main (void) {



        DataStorageClass shift_info;
        {
                do {
                        DataStorageClass setup_info;
                        DriverClassI drivitI(&setup_info);
                        if (RepeatFlag==No) shift_info=setup_info;
                } while (RepeatFlag!=No); // repeat setup if necessary

        }
        shift_info.data_setup(); // clean up some data

        DataInitializationDataPacket * DIDP =new
                DataInitializationDataPacket (shift_info.crew_data,
                shift_info.tides_data,
                shift_info.move_data,
                shift_info.route_data,
                shift_info.vessel_data );
        assert (DIDP!=NULL);
        FerryData_global.insert (DIDP);
        {
                DriverClassM drivitm(shift_info);
                drivitm.run();

        }

        return 0;
}
```

```
.AUTODEPEND

#               *Translator Definitions*
CC = bcc +FERRYLOG.CFG
TASM = TASM
TLIB = tlib
TLINK = tlink
LIBPATH = C:\BORLANDC\LIB;C:\BORLANDC\TVISION\LIB
INCLUDEPATH = C:\BORLANDC\INCLUDE;C:\BORLANDC\TVISION\INCLUDE


#               *Implicit Rules*
.c.obj:
  $(CC) -c {$< }

.cpp.obj:
  $(CC) -c {$< }

#               *List Macros*


EXE_dependencies =  \
 datview.obj \
 flpchk.obj \
 combdat.obj \
 finprt1.obj \
 tmdis.obj \
 pstream.obj \
 misgui1.obj \
 tinputli.obj \
 guiwx.obj \
 ptest.obj \
 mycopy.obj \
 c_func2.obj \
 driver3.obj \
 frrydat2.obj \
 gad.obj \
 guidri2.obj \
 guiini2.obj \
 guimov3.obj \
 myvis2.obj

#               *Explicit Rules*
ferrylog.exe: ferrylog.cfg $(EXE_dependencies)
  $(TLINK) /v/x/c/P-/C/L$(LIBPATH) @&&|
c0l.obj+
datview.obj+
flpchk.obj+
combdat.obj+
finprt1.obj+
tmdis.obj+
pstream.obj+
misgui1.obj+
tinputli.obj+
guiwx.obj+
ptest.obj+
```

```
mycopy.obj+
c_func2.obj+
driver3.obj+
frrydat2.obj+
gad.obj+
guidri2.obj+
guiini2.obj+
guimov3.obj+
myvis2.obj
ferrylog
                # no map file
graphics.lib+
tv.lib+
cl.lib
|


#               *Individual File Dependencies*
datview.obj: ferrylog.cfg datview.cpp

flpchk.obj: ferrylog.cfg flpchk.cpp

combdat.obj: ferrylog.cfg combdat.cpp

finprt1.obj: ferrylog.cfg finprt1.cpp

tmdis.obj: ferrylog.cfg tmdis.cpp

pstream.obj: ferrylog.cfg pstream.cpp

misgui1.obj: ferrylog.cfg misgui1.cpp

tinputli.obj: ferrylog.cfg tinputli.cpp

guiwx.obj: ferrylog.cfg guiwx.cpp

ptest.obj: ferrylog.cfg ptest.cpp

mycopy.obj: ferrylog.cfg mycopy.cpp

c_func2.obj: ferrylog.cfg c_func2.cpp

driver3.obj: ferrylog.cfg driver3.cpp

frrydat2.obj: ferrylog.cfg frrydat2.cpp

gad.obj: ferrylog.cfg gad.cpp

guidri2.obj: ferrylog.cfg guidri2.cpp

guiini2.obj: ferrylog.cfg guiini2.cpp

guimov3.obj: ferrylog.cfg guimov3.cpp

myvis2.obj: ferrylog.cfg myvis2.cpp

#               *Compiler Configuration File*
ferrylog.cfg: ferrylog.mak
 copy &&|
```

```
-ml
-f-
-ff-
-N
-O
-Oe
-Ob
-Va
-Z
-k-
-rd
-vi-
-wbbf
-wpin
-wamb
-wamp
-wasm
-wpro
-wcln
-wdef
-wsig
-wnod
-wstv
-wucp
-weas
-wpre
-I$(INCLUDEPATH)
-L$(LIBPATH)
-P
| ferrylog.cfg
```

```
//this file is finprt1.cpp and is used in the ferrylog program
/* this file contains code for the Summary print feature.  This file
contains a basic parser, and then organizes the data for printout.
This file was adapted from a file used in the combine program */

#include "c_funcr.h"
#include <assert.h>
#include <stdio.h>
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>
#include "consts.h"
#include "frrydat2.h"
#include "combdat.h"
#include <dos.h>
#include <time.h>
// #pragma intrinsic -strncpy
// unsigned _stklen=8192;
#include <tv.h>
const int MAXDATALINE =400;
const int MAXOUTLINELENGTH=500;
const int MAXCOMMANDLENGTH=50;
const int TRUE = 1;
const int FALSE= 0;
FILE *Readfile_ptr;
// string mini class

void code_strip(char *return_code, char *file);
int HandleInitialize(char data_line[MAXDATALINE]);
int HandleMovement (char data_line[MAXDATALINE]);
int HandleIncident (char data_line[MAXDATALINE]);
int HandleDrills (char data_line[MAXDATALINE]);
int HandleWeather (char data_line[MAXDATALINE]);
int BottomCheck(FILE *);
void CanonPrinterSetup(void);
void CanonPrinterCleanup(void);

/* global variables */
extern prnstream PrinterStream; // printer

CombINITDataCollection collection_init=CombINITDataCollection(2,2);
CombMVDataCollection collection_move=CombMVDataCollection(30,5);
CombDRDataCollection collection_drills=CombDRDataCollection(5,2);
CombINCDataCollection collection_incid=CombINCDataCollection(20,3);
CombWxDataCollection collection_weath=CombWxDataCollection(10,3);

int FinalReportPrint(char * readfile)  {
        const char PSP[] ="     " ; // print seperator instead of \t

        // now open the input files


        if ( (Readfile_ptr=fopen(readfile, "r"))==NULL) {
                // error  return invalid code
                return INVALID;
```

```
}

// check each input file for special file terminator
/* make appropo for new use

if (BottomCheck(Readfile_ptr)) {
        fprintf(stdout, "%s is not properly terminated. ",argv[1]);
        fprintf(stdout, "Some data might be lost! \n");
}
*/

rewind(Readfile_ptr);

// now start parsing and loading the data

// do file number one

char read_line[MAXDATALINE];
char pass_line[MAXDATALINE];
strncpy(read_line, " ",MAXDATALINE);
strncpy(pass_line, " ",MAXDATALINE);

char *token;
while ( fgets (read_line, MAXDATALINE, Readfile_ptr)!=NULL) {
        //check string and then hand off information to
        //appropriate subfunction
        int result=0;
        strncpy(pass_line,read_line,MAXDATALINE);
        token=strtok(read_line, "\t");
        if (strcmp (token, FILE_MARKER_BOTTOM)==0); // do nothing

        else if (strcmp (token, FILE_MARKER_TOP)==0); // do nothing
        else if (strcmp (token, ACTION_INITIALIZE)==0){
                result=HandleInitialize(pass_line); /*
                        1 in call indicates infile one */
        }

        else if (strcmp (token, ACTION_MOVEMENT) ==0) {
                result=HandleMovement(pass_line);
        }
        else if (strcmp (token, ACTION_INCIDENTS)==0) {
                result=HandleIncident(pass_line);
        }
        else if (strcmp (token, ACTION_DRILLS)==0) {
                result=HandleDrills(pass_line);
        }
        else if (strcmp(token, ACTION_WEATHER)==0) {
                result=HandleWeather(pass_line);
        }
}


fclose(Readfile_ptr);

DataInitializationDataPacket *DID1=
        (DataInitializationDataPacket *) collection_init.at(0);

time_t time_now;
time_now=time(NULL);
```

```
char stime1[26],stime2[26], nowtime[26];
strncpy(stime1,ctime(&DID1->initialize_time.machineTime),26 );
stime1[24]=' ';
strncpy(nowtime,ctime(&time_now),26);

// now begin to print the data
CanonPrinterSetup();


PrinterStream << "Final Shift Report \n\n"
<< "Vessel:" <<PSP<<DID1->vesselInit.vessel_name
<<PSP <<"\n" << "Report Date:"          << "\n"
<<PSP <<nowtime <<PSP <<"\n";
PrinterStream <<"Shift Started at:" <<PSP<<stime1<<"\n";

PrinterStream << "Initial Route:"<<PSP <<
DID1->routeInit.route_name << PSP <<"\n"
<<"Running From: "<<PSP << DID1->moveInit.depart <<PSP<<"\n" ;
PrinterStream << "To:" <<PSP <<DID1->moveInit.arrival <<"\n";
PrinterStream << "Shift Information: \n\n";

DID1->FinalPrint(); // print the rest of the init data

PrinterStream << "\n\n";
//now print the movement data for end one

PrinterStream<<"Please note: Time information below is in the following format:"
<<"\n" <<"DD:HH:MM:SS\n"
<<"\tWhere DD equals day of the month;\n"
<<"\tHH equals hour of the day;\n"
<<"\tMM equals minute of the hour;\n"
<<"\tSS equals second of the minute.\n";
PrinterStream <<"\n\n";
PrinterStream <<"TimeTable \n";

ccIndex Mcount=0; int move_counter=0;
char Movelines[4][MAXOUTLINELENGTH];
char temp[30];
strncpy (temp, " ", 30);
strncpy(Movelines[0]," ",MAXOUTLINELENGTH);
strncpy(Movelines[1]," ",MAXOUTLINELENGTH);
strncpy(Movelines[2]," ",MAXOUTLINELENGTH);
strncpy(Movelines[3]," ",MAXOUTLINELENGTH);

strcat (Movelines[0], "Type\t");
strcat (Movelines[1], "Name\t");
strcat (Movelines[2], "M Time\t");
strcat (Movelines[3], "G Time\t");

Mcount=collection_move.getCount();
MovementDataPacket *MDP =NULL;

for (move_counter=0; move_counter < Mcount; move_counter++) {

        MDP = (MovementDataPacket *)
                collection_move.at(move_counter);
        if ((strcmp(MDP->point_type,ACTION_DEPART)==0)
                && (move_counter!=0 ) ) {
```

```
                PrinterStream <<  Movelines[0]<<"\n"
                <<Movelines[1] <<"\n"
                << Movelines[2]<<"\n"
                <<Movelines[3] <<"\n\n";
                for (int a_count=0; a_count < 4; a_count++) {
                        strncpy(Movelines[a_count]," ",
                        MAXOUTLINELENGTH);
                }
                strcat (Movelines[0], "Type\t");
                strcat (Movelines[1], "Name\t");
                strcat (Movelines[2], "M Time\t");
                strcat (Movelines[3], "G Time\t");
        }
        strcat(Movelines[0], MDP->point_type);
        strcat(Movelines[0],PSP);
        strcat(Movelines[1],MDP->name);
        strcat(Movelines[1],PSP);
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&MDP->time_data.machineTime));
        strcat(Movelines[2],temp);
        strcat(Movelines[2],PSP);
        strftime(temp, 30, " %d:%H:%M:%S",
                localtime(&MDP->time_data.givenTime));
        strcat(Movelines[3],temp);
        strcat(Movelines[3],PSP);
}
PrinterStream << Movelines[0]<<"\n"<<Movelines[1] <<"\n"
                << Movelines[2]<<"\n"<<Movelines[3] <<"\n\n";

// now put out incident information

PrinterStream << "Incident information:  \n \n";
PrinterStream << "Incident:\tM Time\tG Time\t Comment \n";
int Icount = collection_incid.getCount(), icounter;
for (icounter=0; icounter <Icount; icounter++) {
        IncidentsDataPacket * IDP =(IncidentsDataPacket *)
                collection_incid.at(icounter);

        PrinterStream <<IDP->Incident_name <<PSP;
        strncpy(temp," ",30);
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&IDP->time_data.machineTime));
        PrinterStream << temp<<PSP;
        strncpy(temp," ",30);
        strftime(temp, 30," %d:%H:%M:%S",
                localtime(&IDP->time_data.givenTime));
        PrinterStream << temp<< PSP
        << IDP->comment <<"\n";
}

 // drills information
PrinterStream <<"\n\n" << "Drills information: \n";
PrinterStream << "drill:\tM Time\tG Time\t Comment \n";
int Dcount = collection_drills.getCount(), dcounter;
for (dcounter=0; dcounter <Dcount; dcounter++) {
        DrillsDataPacket * DDP =(DrillsDataPacket *)
                collection_drills.at(dcounter);

        PrinterStream <<DDP->drill_name <<PSP;
```

```
else if (strcmp(tokenB, INIT_KEYWD_ROUTE)==0) {
        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.route_data.route_name, tokenC,
                MaxRouteNameLength);
        // insert initial route info
}
else if (strcmp(tokenB, INIT_KEYWD_WATCH)==0) {
        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.crew_data.watch_name,tokenC,2);
        // insert watch letter
}
else if (strcmp(tokenB, INIT_KEYWD_CREW)==0) {
        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.crew_data.crew_position_name[crew_counter],
                tokenC,crew_name_length);
        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.crew_data.crew_name[crew_counter],
                tokenC,crew_name_length);
        crew_counter++;
        // put in crew position and persons name
}
else if (strcmp(tokenB, INIT_KEYWD_TIDE)==0) {
        char *junk;
        junk=strtok(NULL,"\t"); // junk high
        junk=strtok(NULL,"\t"); // junk height
        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.tides_data.high[tides_counter].height,
                tokenC, tides_length);
        junk=strtok(NULL,"\t"); // junk time
        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.tides_data.high[tides_counter].time,
                tokenC, tides_length);
        junk=strtok(NULL,"\t"); // junk low
        junk=strtok(NULL,"\t"); // junk height

        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.tides_data.low[tides_counter].height,
                tokenC, tides_length);

        junk=strtok(NULL,"\t"); // junk time

        tokenC=strtok(NULL,"\t");
        strncpy(InitStuff.tides_data.low[tides_counter].time,
                tokenC, tides_length);
        tides_counter++;
        // read in tides information

}
else if (strcmp(tokenB, INIT_KEYWD_DONE)==0) {
        // now pass this structure off to the DIDP

        DataInitializationDataPacket * DIDP =new
        DataInitializationDataPacket (InitStuff.crew_data,
        InitStuff.tides_data,
        InitStuff.move_data,
        InitStuff.route_data,
        InitStuff.vessel_data );
        assert (DIDP!=NULL);
        DIDP->initialize_time=InitTime ;
```

```
                    //FerryData_global.insert (DIDP);
                    collection_init.insert(DIDP);
                }

        //      else printf("Unknown entry %s \n", tokenB);


                return 0;
        }
        int HandleMovement (char data_line[MAXDATALINE]){

                OccurTime move_time;
                char point_name[MAXCOMMANDLENGTH],
point_type[MAXCOMMANDLENGTH];
                char *token;
                token=strtok(data_line,"\t");  // MOVEMENT
                token=strtok(NULL,"\t"); // point name
                strncpy (point_name, token,MAXCOMMANDLENGTH);
                token=strtok(NULL,"\t"); // point type
                strncpy(point_type, token,MAXCOMMANDLENGTH);
                token=strtok(NULL,"\t"); // machineTime
                move_time.machineTime=atol(token);
                token=strtok(NULL,"\t"); // givenTime
                move_time.givenTime=atol(token);
                MovementDataPacket *MDP = new
                        MovementDataPacket(point_name, point_type,move_time);
                collection_move.insert(MDP);

                return 0;
        }
        int HandleIncident (char data_line[MAXDATALINE]){

                OccurTime incident_time;
                char incident_name[MAXCOMMANDLENGTH], *token,
                        comment[MaxCommandLength];

                token=strtok(data_line,"\t");  // INCIDENT
                token=strtok(NULL,"\t"); // incident name
                strncpy (incident_name, token,MAXCOMMANDLENGTH);
                token=strtok(NULL,"\t"); // machineTime
                incident_time.machineTime=atol(token);
                token=strtok(NULL,"\t"); // givenTime
                incident_time.givenTime=atol(token);
                token=strtok(NULL,"\t"); // comment
        //      mstrncpy(comment, token,strlen(token)+2);   // problem is here
                IncidentsDataPacket *IDP = new
                        IncidentsDataPacket(incident_name, incident_time,token);
        //              IncidentsDataPacket(incident_name, incident_time,comment);
                /* incident_name gets overwritten by the call to strncpy */
                collection_incid.insert(IDP);
                return 0;

        }
        int HandleDrills (char data_line[MAXDATALINE]){
                OccurTime drill_time;
                char drill_name[MAXCOMMANDLENGTH],
 *token,comment[MaxCommentLength];
                token=strtok(data_line,"\t");  // DRILL
                token=strtok(NULL,"\t"); // drill name
```

```
        strncpy (drill_name, token,MAXCOMMANDLENGTH);
        token=strtok(NULL,"\t"); // machineTime
        drill_time.machineTime=atol(token);
        token=strtok(NULL,"\t"); // givenTime
        drill_time.givenTime=atol(token);
        token=strtok(NULL,"\t"); // comment
//      strncpy(comment, token,MaxCommentLength);
        DrillsDataPacket *DDP = new
                DrillsDataPacket(drill_name, drill_time,token);
        collection_drills.insert(DDP);


        return 0;
}

// weather
int HandleWeather (char data_line[MAXDATALINE]){

        OccurTime weatherTime;
        WeatherInfo weatherData;
        char  *token;


        token=strtok(data_line,"\t");  // WEATHER
        token=strtok(NULL,"\t"); // clouds
        strncpy (weatherData.Clouds, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // precip
        strncpy (weatherData.Precip, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // visability
        strncpy (weatherData.Vis, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // wind direction
        strncpy (weatherData.WindDir, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // wind speed
        strncpy (weatherData.WindSpeed, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // sea
        strncpy (weatherData.Sea, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // temp
        strncpy (weatherData.Temp, token,MaxWeatherLength);
        token=strtok(NULL,"\t"); // barometer
        strncpy (weatherData.Barom, token,MaxWeatherLength);

        token=strtok(NULL,"\t"); // machineTime
        weatherTime.machineTime=atol(token);
        token=strtok(NULL,"\t"); // givenTime
        weatherTime.givenTime=atol(token);

        WeatherDataPacket *WDP= new
                WeatherDataPacket (weatherData,weatherTime);
        collection_weath.insert(WDP);

        return 0;

}
void CanonPrinterSetup(void) {
        char CondensedMode =0x0F;
        PrinterStream.init();
        PrinterStream << CondensedMode;
        PrinterStream.flush();
}
```

```
void CanonPrinterCleanup(void) {
        char twlvePtMode=0x12;
        PrinterStream << "\f" << twlvePtMode;
        PrinterStream.flush();
}
```

```
// subroutine to check the floppy drive for a disk
// checks for formatting and sufficient space
// this file is used to build the ferrylog program
  // this file is flpchk.cpp


#include <bios.h>
#include <string.h>
#include <dos.h>
#include <iostream.h>
#include <sys\stat.h>
#include "consts.h"
#include "miscdat.h"

const int FlpOk= 0x00 ;
const int FlpBadCommand=0x01  ;
const int FlpAddMark= 0x02; // unformatted or other error
const int FlpWrPr= 0x03; // write protected disk
const int FlpSect= 0x04;
const int FlpReset= 0x05;
const int FlpDiskCh= 0x06;
const int FlpDrPar= 0x07;
const int FlpDMAOv= 0x08;
const int FlpDmaAc= 0x09;
const int FlpDmaBadSec= 0x0A;
const int FlpBadTr= 0x0B;
const int FlpUnTr= 0x0C;
const int FlpCrcErr= 0x10;
const int FlpCrcFix= 0x11;
const int FlpConFail= 0x20;
const int FlpSkFail= 0x40;
const int FlpAtFail= 0x80; //Drive Not Ready
const int FlpHardNR= 0xAA;
const int FlpHardUd= 0xBB;
const int FlpWrtFlt= 0xCC;
const int FlpStErr= 0xE0;
const int FlpSnFail= 0xFF;


FlopInfo FlopCheck (char * FileName) {
        int DiskCheck (void) ;
        long FileSize (char *);
        struct dfree diskFree;
        long avail, Fsize;
        const int ADrive=1;

        FlopInfo returnInfo;
        // Initialize floppy info structure
        strncpy (returnInfo.FlopStatusTxt, " ", MaxReturnLength);
        returnInfo.FlopStatusInt=0;
        // first test floppy Drive for readability

        int DiskStatus=DiskCheck();
        if (DiskStatus==FlpOk) {
                // check for available space
```

```
        getdfree(ADrive, &diskFree);
        if(diskFree.df_sclus==0xFFFF) {
                returnInfo.FlopStatusInt=(DISK_INVALID); // disk error
                strncpy(returnInfo.FlopStatusTxt,
                        "Replace disk in drive A: with a fresh one",
                        MaxReturnLength);

                return returnInfo;
        }
        else {
                avail =(long) diskFree.df_avail
                        *(long) diskFree.df_bsec
                        *(long) diskFree.df_sclus;
        }
        Fsize= FileSize(FileName);
        if ((Fsize!=(-1L) ) &&(Fsize< avail) ) {
                returnInfo.FlopStatusInt=(DISK_VALID); // OK
                strncpy(returnInfo.FlopStatusTxt,
                        "ok",MaxReturnLength);
                return returnInfo;
        }
        else if (Fsize==(-1L)) {
                // file not found (probably)
                strncpy(returnInfo.FlopStatusTxt,
                "File to be copied not found",
                MaxReturnLength);
                returnInfo.FlopStatusInt=(DISK_INVALID);
                return returnInfo;
        }
        else {
                strncpy(returnInfo.FlopStatusTxt,
                        "Not enough Space on Floppy Disk: Replace",
                        MaxReturnLength);
                returnInfo.FlopStatusInt=(DISK_INVALID);
        }
}
else {  // problem with the disk
        returnInfo.FlopStatusInt=(DISK_INVALID);
        switch (DiskStatus) {
                case FlpAddMark:
                        strncpy(returnInfo.FlopStatusTxt,
                                "Floppy Error. Replace disk",
                                MaxReturnLength);
                        break;
                case FlpWrPr:
                        strncpy(returnInfo.FlopStatusTxt,
                                "Write Protect error. Fix",
                                MaxReturnLength);
                        break;
                case FlpAtFail:
                        strncpy(returnInfo.FlopStatusTxt,
                                "Put disk in drive A:",
                                MaxReturnLength);
                        break;
                default:
                        strncpy(returnInfo.FlopStatusTxt,
                                "Drive A error. Fix",
                                MaxReturnLength);
                        break;
```

```
                }

        }

        return returnInfo;
}
long FileSize(char * Fname) {
        struct stat statinfo;
        long returnValue;

        if (stat(Fname, &statinfo)!=0) {
                returnValue=(-1L); //return problem with disk
        }
        else returnValue=statinfo.st_size;

        return returnValue;
}

int DiskCheck(void) {
        static int call_counter=0;
        const int ResetCMD=0;
        const int ReadCMD=2; // read  sector
        const int WriteCMD=3;  // write sector
                //Used to check for Write Protection
        const int DRIVE=0; // Drive a
        const int HEAD=0; // disk head #
        const int TRACK=1; // DISK TRACK #
        const int SECT =1; // sector number
        const int NSECT=1; // sector count

        int BiosResult;
        char buffer [512] ;
        strncpy(buffer, " ", 512);


        // if call counter > 0, then system is trying to write to the
        // disk after an inital failure.  On generic pc, drive did
        // odd things if drive door was opened and then shut
        // try a disk controller reset to try to alleviate the problem

        if (call_counter > 0) // reset disk
        BiosResult=biosdisk (ResetCMD,DRIVE,HEAD,TRACK,SECT,NSECT,buffer);
        call_counter++;
        //check first time
        BiosResult=biosdisk (ReadCMD,DRIVE,HEAD,TRACK,SECT,NSECT,buffer);
        // check second time
        if(BiosResult!=0)
                BiosResult=biosdisk
(ReadCMD,DRIVE,HEAD,TRACK,SECT,NSECT,buffer);
        if (BiosResult==0) // check for writeProtect

BiosResult=biosdisk(WriteCMD,DRIVE,HEAD,TRACK,SECT,NSECT,buffer);

        // two checks needed for biosdisk (media change)

        return BiosResult;
}
```

```
// this file is frrydat2.cpp and is used by both the Combine and Ferrylog
//programs
/* offending code to either the combine or ferrylog programs are selected
out using #define and #if statements
  This is why a frrydat2.obj object file compiled for the combine
  program is not link compatible with the ferrylog log program
  and vice versa.  To overcome this, if using a make file, delete
  this frrydat2.obj file (or all .obj files) before building
  the other program.  Or, if using the Borland IDE, choose
  the Build All option from the compile menu rather than the Make option

  */

/* this file contains a number of class member print routines, actually,
most if not all of the routines needed for printing are in this file
These routines could be combined and condensed, especially if judicious
use of templates were used.
*/

// this gives the class body for non-gui classes
// data Storage Class body
#include <string.h>
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include "frrydat2.h"
#include "misgui1.h"
//#include "consts.h"
#include <time.h>

// globals
#ifndef COMBINE
extern PrintOptionData POD;
extern fstream VisFile; //read and append modes
extern prnstream PrinterStream;
extern FerryFileNames outfiles; // names for the data output files
                                // defined in the driver module
#endif  // define only for ferry log, not for the combination program
DataStorageClass::DataStorageClass(void) {


        setup_flag=0;  // flag that data has not been modified after
                       // initial setup


        // initialize the crew information to zip
        for (int count1=0; count1 < crew_positions; count1++) {
                for (int count2=0; count2< crew_name_length; count2++) {
                        crew_data.crew_position_name[count1][count2]='\0';
                        crew_data.crew_name[count1][count2]='\0';
                }
                crew_data.crew_name[count1][0]=' ';
        }
        strncpy (crew_data.crew_position_name[0], CREW_POS_0_NAME,
crew_name_length);
```

```
                strncpy (crew_data.crew_position_name[1], CREW_POS_1_NAME,
crew_name_length);
                strncpy (crew_data.crew_position_name[2], CREW_POS_2_NAME,
crew_name_length);
                strncpy (crew_data.crew_position_name[3], CREW_POS_3_NAME,
crew_name_length);
                strncpy (crew_data.crew_position_name[4], CREW_POS_4_NAME,
crew_name_length);
                strncpy (crew_data.crew_position_name[5], CREW_POS_5_NAME,
crew_name_length);
                strncpy (crew_data.crew_position_name[6], CREW_POS_6_NAME,
crew_name_length);
                strncpy (crew_data.crew_position_name[7], CREW_POS_7_NAME,
crew_name_length);
                strncpy (crew_data.crew_position_name[8], CREW_POS_8_NAME,
crew_name_length);
                strncpy (crew_data.crew_position_name[9], CREW_POS_9_NAME,
crew_name_length);
                strncpy (crew_data.crew_position_name[10], CREW_POS_10_NAME,
crew_name_length);



                // initialize the tides information to zip

                for (count1=0; count1 < tides_entries; count1++) {
                        for (int count2=0; count2 < tides_length; count2++) {

                                tides_data.high[count1].height[count2]='\0';
                                tides_data.high[count1].time[count2]='\0';

                                tides_data.low[count1].height[count2]='\0';
                                tides_data.low[count1].time[count2]='\0';

                        }
                        tides_data.high[count1].height[0]=' ';
                        tides_data.high[count1].time[0]=' ';

                        tides_data.low[count1].height[0]=' ';
                        tides_data.low[count1].time[0]=' ';


                }

                // initialize the route information to zip
                for (count1=0; count1<MaxRouteNameLength; count1++)
                        route_data.route_name[count1]='\0';

                for (int count2=0; count2 < MaxCheckLength; count2++) {
                        for (count1=0; count1 < MaxRouteData_ad; count1++) {
                                route_data.arrival[count1][count2]='\0';
                                route_data.depart[count1][count2]='\0';
                        }
                        for (count1=0; count1 <MaxRouteData_ck; count1++) {
                                route_data.chkpoint[count1][count2]='\0';
                        }
                }
                // initialize the vessel information to zip
                for (count1=0; count1< MaxVesselNameLength; count1++) {
```

```
                vessel_data.vessel_name[count1]='\0';
        }

// initialize move information to zip
        for (int i=0; i < MaxCheckLength; i++) {
                move_data.arrival[i]='\0';
                move_data.depart[i]='\0';
                for (int j=0; j< MaxCheckPoints-2; j++) {
                        move_data.chkpoint[j][i]='\0';
                }
        }

}
void DataStorageClass::data_setup (void) {


        // clean crew_data.crew_name


        int Ccounter=0, Ecounter=0;
        for (Ecounter=0; Ecounter <crew_positions; Ecounter++) {
                for (Ccounter =0; Ccounter < crew_name_length; Ccounter++) {
                   if (crew_data.crew_name[Ecounter][Ccounter]=='\0') break;
                }
                for (Ccounter++; Ccounter < crew_name_length; Ccounter++ )
                        crew_data.crew_name[Ecounter][Ccounter]='\0';
        }

        // fix tides
        for (Ecounter=0; Ecounter < tides_entries; Ecounter ++ ) {

                for (Ccounter =0; Ccounter < tides_length; Ccounter++)
                        if (tides_data.high[Ecounter].height[Ccounter]=='\0')
                                break;
                for (Ccounter++; Ccounter < tides_length; Ccounter++ )
                        tides_data.high[Ecounter].height[Ccounter]='\0';

                for (Ccounter =0; Ccounter < tides_length; Ccounter++)
                        if (tides_data.high[Ecounter].time[Ccounter]=='\0')
                                break;
                for (Ccounter++; Ccounter < tides_length; Ccounter++ )
                        tides_data.high[Ecounter].time[Ccounter]='\0';

                // low tides

                for (Ccounter =0; Ccounter < tides_length; Ccounter++)
                        if (tides_data.low[Ecounter].height[Ccounter]=='\0')
                                break;
                for (Ccounter++; Ccounter < tides_length; Ccounter++ )
                        tides_data.low[Ecounter].height[Ccounter]='\0';

                for (Ccounter =0; Ccounter < tides_length; Ccounter++)
                        if (tides_data.low[Ecounter].time[Ccounter]=='\0')
                                break;
                for (Ccounter++; Ccounter < tides_length; Ccounter++ )
                        tides_data.low[Ecounter].time[Ccounter]='\0';
        }

        // now check for null in first character (bad!)
```

```
        for (Ecounter=0; Ecounter < tides_entries; Ecounter ++ ) {
                if (tides_data.high[Ecounter].height[0]=='\0')
                        tides_data.high[Ecounter].height[0]=' ';
                if (tides_data.high[Ecounter].time[0]=='\0')
                        tides_data.high[Ecounter].time[0]=' ';
                if (tides_data.low[Ecounter].height[0]=='\0')
                        tides_data.low[Ecounter].height[0]=' ';
                if (tides_data.low[Ecounter].time[0]=='\0')
                        tides_data.low[Ecounter].time[0]=' ';
        }
        //move_setup();
        setup_flag=1; // change flag to indicate data cleaned

}
void DataStorageClass:: route_setup(void) {
        // setup the possible arrive and depart info for the given route
        if (strcmp (route_data.route_name, ROUTE_NAME_SEAW)==0) {
                strncpy(route_data.arrival[0], "Seattle", MaxCheckLength);
                strncpy(route_data.arrival[1], "Winslow", MaxCheckLength);
                strncpy(route_data.depart[0], "Seattle", MaxCheckLength);
                strncpy(route_data.depart[1], "Winslow", MaxCheckLength);

        }
        else {
                int counter=0;
                for (counter=0; counter < MaxRouteData_ad; counter++) {
                        strncpy(route_data.arrival[counter], "Blank",
                                MaxCheckLength);
                        strncpy(route_data.depart[counter], "Blank",
                                MaxCheckLength);
                }
        }

}
void DataStorageClass::move_setup(void) {


        // now setup movement button names
        if (strcmp(route_data.route_name, ROUTE_NAME_SEAW)==0) {


                if (strcmp(move_data.depart,"Winslow")==0) {
                        move_data.number_of_points=6;
                        strncpy(move_data.chkpoint[0],"Five",MaxCheckLength);
                        strncpy(move_data.chkpoint[1],"Three", MaxCheckLength);
                        strncpy(move_data.chkpoint[2],"Tyee", MaxCheckLength);
                        strncpy(move_data.chkpoint[3],"Duam", MaxCheckLength);
                        strncpy(move_data.chkpoint[4],"Half", MaxCheckLength);
                        strncpy(move_data.chkpoint[5],"Slow", MaxCheckLength);
                }
                else {
                        move_data.number_of_points=5;
                        strncpy(move_data.chkpoint[5]," ",MaxCheckLength);
                        strncpy(move_data.chkpoint[4],"Five", MaxCheckLength);
                        strncpy(move_data.chkpoint[3],"Three", MaxCheckLength);
                        strncpy(move_data.chkpoint[2],"Tyee", MaxCheckLength);
                        strncpy(move_data.chkpoint[1],"C/Cr", MaxCheckLength);
                        strncpy(move_data.chkpoint[0],"Duam", MaxCheckLength);
                }
```

```
        }
        else {  // expand later for additional routes
                move_data.number_of_points=4;
//              strncpy(move_data.arrival, "Generic", MaxCheckLength);
//              strncpy(move_data.depart, "Generic", MaxCheckLength);
                strncpy(move_data.chkpoint[0],"Generic",MaxCheckLength);
                strncpy(move_data.chkpoint[1],"Generic", MaxCheckLength);
                strncpy(move_data.chkpoint[2],"Generic", MaxCheckLength);
                strncpy(move_data.chkpoint[3],"Generic", MaxCheckLength);
        }
}
/* information packets to communicate with the TNSCollection object */


DataInitializationDataPacket::DataInitializationDataPacket
        (CrewInfo crewPass, TidesInfo tidesPass,
        MoveInfo movePass, RouteInfo routePass, VesselInfo vesselPass) {


        action_id=newStr(ACTION_INITIALIZE);
        crewInit=crewPass;
        tidesInit=tidesPass;
        moveInit=movePass;
        routeInit=routePass;
        vesselInit=vesselPass;
        // setup time
        initialize_time.machineTime=initialize_time.givenTime=time(NULL);
}
DataInitializationDataPacket::~DataInitializationDataPacket(void) {
        delete (char *) action_id;
}
void DataInitializationDataPacket::VisPrint(fstream *Vs) {
        int counter;
        char Temp [30];
        strncpy (Temp, " ",30);
        //strftime(Temp, 30," %d:%H:%M:%S",
        //                localtime(&initialize_time.givenTime));
        strcpy (Temp, ctime(&initialize_time.givenTime));
        *Vs << "Time and Date:\n"<< Temp <<"\n";
        *Vs << "Vessel:\t"<<vesselInit.vessel_name <<"\n";
        *Vs << "End:\t"<<vesselInit.vessel_end <<"\n";
        *Vs << "Route:\t" <<routeInit.route_name<< "\n";

        // write out the Ferry information

        *Vs<< "Watch: \t"
        << crewInit.watch_name <<"\n\n"
        <<"Crew:\n";
        for (counter=0; counter < crew_positions; counter++) {
                *Vs << setiosflags( ios::left )
                << crewInit.crew_position_name[counter] <<"\t"
                << crewInit.crew_name[counter] <<"\n";
        }
        *Vs <<"\n";
        // tides info
        *Vs << "Tides\tHeight\tTime\n";

        for (counter=0; counter < tides_entries ; counter++) {
```

```
                *Vs << setiosflags( ios::left )
                <<"High"<<counter<<"\t"
                << tidesInit.high[counter].height
                << "\t"
                << tidesInit.high[counter].time
                << "\n";
        }
        for (counter=0; counter < tides_entries ; counter++) {
                *Vs<<"Low"<<counter<<"\t"
                << tidesInit.low[counter].height
                << "\t"
                << tidesInit.low[counter].time
                << "\n";
        }
        *Vs<<"\n\n";


}
void DataInitializationDataPacket::SpewPrint(prnstream *es) {
        int counter;
        char Temp [30];
        strncpy (Temp, " ",30);
        //strftime(Temp, 30," %d:%H:%M:%S",
        //                localtime(&initialize_time.givenTime));
        strcpy (Temp, ctime(&initialize_time.givenTime));
        *es << "Time and Date:\n"<< Temp <<"\n";
        *es << "Vessel:\t"<<vesselInit.vessel_name <<"\n";
        *es << "End:\t"<<vesselInit.vessel_end <<"\n";
        *es << "Route:\t" <<routeInit.route_name<< "\n";

        // write out the Ferry information

        *es<< "Watch: \t"
        << crewInit.watch_name <<"\n\n"
        <<"Crew:\n";
        for (counter=0; counter < crew_positions; counter++) {
                *es << setiosflags( ios::left )
                << crewInit.crew_position_name[counter] <<"\t"
                << crewInit.crew_name[counter] <<"\n";
        }
        *es <<"\n";
        // tides info
        *es << "Tides\tHeight\tTime\n";

        for (counter=0; counter < tides_entries ; counter++) {
                *es << setiosflags( ios::left )
                <<"High"<<counter<<"\t"
                << tidesInit.high[counter].height
                << "\t"
                << tidesInit.high[counter].time
                << "\n";
        }
        for (counter=0; counter < tides_entries ; counter++) {
                *es<<"Low"<<counter<<"\t"
                << tidesInit.low[counter].height
                << "\t"
                << tidesInit.low[counter].time
                << "\n";
        }
```

```
        *es<<"\n\n";

}
void DataInitializationDataPacket::ExcPrint(ofstream * es) {
        // put print info here
        int counter;
        *es<< "Watch: \t"
        << crewInit.watch_name <<"\n\n"
        <<"Crew:\n";
        for (counter=0; counter < crew_positions; counter++) {
                *es << setiosflags( ios::left )
                << crewInit.crew_position_name[counter] <<"\t"
                << crewInit.crew_name[counter] <<"\n";
        }
        *es <<"\n";
        // tides info
        *es << "Tides\tHeight\tTime\n";

        for (counter=0; counter < tides_entries ; counter++) {
                *es << setiosflags( ios::left )
                <<"High"<<counter<<"\t"
                << tidesInit.high[counter].height
                << "\t"
                << tidesInit.high[counter].time
                << "\n";
        }
        for (counter=0; counter < tides_entries ; counter++) {
                *es<<"Low"<<counter<<"\t"
                << tidesInit.low[counter].height
                << "\t"
                << tidesInit.low[counter].time
                << "\n";
        }
        *es<<"\n\n";

}
#ifndef COMBINE
void DataInitializationDataPacket::FinalPrint(void) {
        // put print info here
        int counter;
        PrinterStream<< "Watch: \t"
        << crewInit.watch_name <<"\n\n"
        <<"Crew:\n";
        for (counter=0; counter < crew_positions; counter++) {
                PrinterStream << setiosflags( ios::left )
                << crewInit.crew_position_name[counter] <<"\t"
                << crewInit.crew_name[counter] <<"\n";
        }
        PrinterStream <<"\n";
        // tides info
        PrinterStream << "Tides\tHeight\tTime\n";

        for (counter=0; counter < tides_entries ; counter++) {
                PrinterStream << setiosflags( ios::left )
                <<"High"<<counter<<"\t"
                << tidesInit.high[counter].height
                << "\t"
                << tidesInit.high[counter].time
                << "\n";
```

```
        }
        for (counter=0; counter < tides_entries ; counter++) {
                PrinterStream<<"Low"<<counter<<"\t"
                << tidesInit.low[counter].height
                << "\t"
                << tidesInit.low[counter].time
                << "\n";
        }
        PrinterStream<<"\n\n";

}
#endif
void DataInitializationDataPacket::write(ofstream * os) {
        // print out the initialization data
        int counter=0;

        // vessel initialization

        // name
        *os << setiosflags(ios::left)
        << action_id << "\t" << INIT_KEYWD_VES_NAME<<"\t"
        << vesselInit.vessel_name <<"\t"
        << initialize_time.machineTime <<"\t"
        << initialize_time.givenTime <<"\t"<<"\n";

        // end  0= only 1= one 2=two

        *os << setiosflags(ios::left)
        << action_id << "\t" << INIT_KEYWD_VES_END<<"\t"
        << vesselInit.vessel_end <<"\t"
        << initialize_time.machineTime <<"\t"
        << initialize_time.givenTime <<"\t" <<"\n";

        //Route Info
        *os << setiosflags(ios::left)
        << action_id << "\t" << INIT_KEYWD_ROUTE << "\t"
        << routeInit.route_name << "\t"
        << initialize_time.machineTime <<"\t"
        << initialize_time.givenTime <<"\t" <<"\n";

        // Initial arrival information
        *os << setiosflags(ios::left)
        << action_id << "\t" << INIT_KEYWD_ARR<<"\t"
        << moveInit.arrival <<"\t"   // might want to change to route str
        << initialize_time.machineTime <<"\t"
        << initialize_time.givenTime <<"\t" <<"\n";

        // initial departure information
        *os << setiosflags(ios::left)
        << action_id << "\t" << INIT_KEYWD_DEP<<"\t"
        << moveInit.depart <<"\t"  // might want to change to route str
        << initialize_time.machineTime <<"\t"
        << initialize_time.givenTime <<"\t" <<"\n";

        // watch writeout

        *os << setiosflags(ios::left)
        << action_id << "\t" << INIT_KEYWD_WATCH<<"\t"
        << crewInit.watch_name <<"\t"
```

```
        << initialize_time.machineTime <<"\t"
        << initialize_time.givenTime <<"\t" <<"\n";

        // crew writeout
        for (counter=0; counter < crew_positions; counter++) {
                *os << setiosflags( ios::left )
                << action_id << "\t" << INIT_KEYWD_CREW<<"\t"
                << crewInit.crew_position_name[counter] <<"\t"
                << crewInit.crew_name[counter] <<"\t"
                << initialize_time.machineTime <<"\t"
                << initialize_time.givenTime <<"\t" <<"\n";
        }
        // tides writeout
        for (counter=0; counter < tides_entries ; counter++) {
                *os << setiosflags( ios::left )
                << action_id << "\t" << INIT_KEYWD_TIDE<<"\t"
                << INIT_KEYWD_TIDEh<<"\t" << INIT_KEYWD_TIHI<<"\t"
                << tidesInit.high[counter].height
                << "\t"
                << INIT_KEYWD_TITM<<"\t"
                << tidesInit.high[counter].time
                << "\t"
                << INIT_KEYWD_TIDEl<<"\t" << INIT_KEYWD_TIHI<<"\t"
                << tidesInit.low[counter].height
                << "\t"
                << INIT_KEYWD_TITM<<"\t"
                << tidesInit.low[counter].time
                << "\t"
                << initialize_time.machineTime <<"\t"
                << initialize_time.givenTime <<"\t" <<"\n";
                // concludes output of initstuff: state so in file

        }
        *os<<action_id << "\t" << INIT_KEYWD_DONE <<"\t\t\n";


}


MovementDataPacket::MovementDataPacket (char *pass_name, char *pass_type,
        OccurTime pass_time) {

        // initialize the data elements
        action_id=newStr(ACTION_MOVEMENT);
        name=newStr(pass_name); // TVison global string copier
        point_type=newStr(pass_type);
        time_data=pass_time;
}
MovementDataPacket::~MovementDataPacket(void) {
        delete (char *) action_id;
        delete (char *) name;
        delete (char *) point_type;
}
void MovementDataPacket::write (ofstream *os) {
        *os << setiosflags( ios::left )
        << action_id
        << "\t" << name
        << "\t" << point_type
```

```
        << "\t" << time_data.machineTime
        << "\t" << time_data.givenTime
        << "\t\n";
}
void MovementDataPacket::VisPrint(fstream *Vs) {

        char temp [30];
        strftime(temp, 30," %d:%H:%M:%S",
                        localtime(&time_data.givenTime));

        *Vs << setiosflags( ios::left )
        << action_id
        << "\t" << name
        << "\t" << point_type
        << "\t" << temp  // flush??
        << "\t\n";

}
void MovementDataPacket::SpewPrint(prnstream *os) {

        char temp [30];
        strftime(temp, 30," %d:%H:%M:%S",
                        localtime(&time_data.givenTime));

        *os << setiosflags( ios::left )
        << action_id
        << "\t" << name
        << "\t" << point_type
        << "\t" << temp  // flush??
        << "\t\n";
}

// DrillsDataPacket

DrillsDataPacket::DrillsDataPacket (char *pass_drill, OccurTime pass_time,
        char * pass_comment) {

        // initialize the data elements
//      strcpy(action_id, ACTION_Drills);
        action_id=newStr(ACTION_DRILLS);
        drill_name=newStr(pass_drill); // TVison global string copier
        comment=newStr(pass_comment);
        time_data=pass_time;
}
DrillsDataPacket::~DrillsDataPacket(void) {
        delete (char *) action_id;
        delete (char *) drill_name;
        delete (char *) comment;

}
void DrillsDataPacket::write (ofstream *os) {
        *os << setiosflags( ios::left )
        << action_id
        << "\t" << drill_name
        << "\t" << time_data.machineTime
        << "\t" << time_data.givenTime
        << "\t" << comment <<"\t"
        << "\n";
}
```

```
void DrillsDataPacket::VisPrint (fstream *Vs) {
char temp [30];
        strftime(temp, 30," %d:%H:%M:%S",
                        localtime(&time_data.givenTime));

        *Vs << setiosflags( ios::left )
        << action_id
        << "\t" << drill_name
        //<< "\t" << time_data.givenTime
        << temp <<"\t"
        << "\n" << comment <<"\t"
        << "\n";


}
void DrillsDataPacket::SpewPrint (prnstream *os) {
        char temp [30];
        strftime(temp, 30," %d:%H:%M:%S",
                        localtime(&time_data.givenTime));

        *os << setiosflags( ios::left )
        << action_id
        << "\t" << drill_name
        //<< "\t" << time_data.givenTime
        << temp <<"\t"
        << "\n" << comment <<"\t"
        << "\n";


}


// IncidentsDataPacket

IncidentsDataPacket::IncidentsDataPacket (char *pass_incident,
        OccurTime pass_time, char * pass_comment) {

        // initialize the data elements

        action_id=newStr(ACTION_INCIDENTS);
        Incident_name=newStr(pass_incident); // TVison global string copier
        comment=newStr(pass_comment);
        time_data=pass_time;
}
IncidentsDataPacket::~IncidentsDataPacket(void) {
        delete (char *) action_id;
        delete (char *) Incident_name;
        delete (char *) comment;

}
void IncidentsDataPacket::write (ofstream *os) {
        *os << setiosflags( ios::left )
        << action_id
        << "\t" << Incident_name
        << "\t" << time_data.machineTime
        << "\t" << time_data.givenTime
        << "\t" << comment <<"\t"
        << "\n";
```

```
}
void IncidentsDataPacket::VisPrint(fstream *Vs) {
        char temp [30];
        strftime(temp, 30," %d:%H:%M:%S",
                        localtime(&time_data.givenTime));
        *Vs << setiosflags( ios::left )
        << action_id
        << "\t" << Incident_name
        << "\t" << temp
        << "\n" << comment <<"\t"
        << "\n";
}
void IncidentsDataPacket::SpewPrint(prnstream *os) {

        char temp [30];
        strftime(temp, 30," %d:%H:%M:%S",
                        localtime(&time_data.givenTime));
        *os << setiosflags( ios::left )
        << action_id
        << "\t" << Incident_name
        << "\t" << temp
        << "\n" << comment <<"\t"
        << "\n";
}

// WeatherInfo class things
// constructor

WeatherInfo::WeatherInfo(void) {
        strncpy (Clouds, " ", MaxWeatherLength);
        strncpy (Precip, " ", MaxWeatherLength);
        strncpy (Vis, " ", MaxWeatherLength);
        strncpy (WindDir, " ", MaxWeatherLength);
        strncpy (WindSpeed, " ", MaxWeatherLength);
        strncpy (Sea, " ", MaxWeatherLength);
        strncpy (Temp, " ", MaxWeatherLength);
        strncpy (Barom, " ", MaxWeatherLength);
}
// weather data packet for collection
WeatherDataPacket::WeatherDataPacket(WeatherInfo &weatherPass,
        OccurTime &timePass) {
        weatherTime=timePass;
        action_id=newStr(ACTION_WEATHER);
        weatherData=weatherPass;


}
void WeatherDataPacket::write(ofstream *os) {

        *os << setiosflags( ios::left )
        << action_id
        << "\t" << weatherData.Clouds
        << "\t" << weatherData.Precip
        << "\t" << weatherData.Vis
        << "\t" << weatherData.WindDir
        << "\t" << weatherData.WindSpeed
        << "\t" << weatherData.Sea
        << "\t" << weatherData.Temp
        << "\t" << weatherData.Barom
```

```
        << "\t" << weatherTime.machineTime
        << "\t" << weatherTime.givenTime
        << "\n";
}
void WeatherDataPacket::VisPrint(fstream *Vs) {
        char temp [30];
        strftime(temp, 30," %d:%H:%M:%S",
                        localtime(&weatherTime.givenTime));

        *Vs << setiosflags( ios::left )
        << action_id
        << "\t" << weatherData.Clouds
        << "\t" << weatherData.Precip
        << "\t" << weatherData.Vis
        << "\t" << weatherData.WindDir
        << "\t" << weatherData.WindSpeed
        << "\t" << weatherData.Sea
        << "\t" << weatherData.Temp
        << "\t" << weatherData.Barom
        << "\t" << temp<<"\t"
        << "\n";
}

void WeatherDataPacket::SpewPrint(prnstream *os) {

        char temp [30];
        strftime(temp, 30," %d:%H:%M:%S",
                        localtime(&weatherTime.givenTime));

        *os << setiosflags( ios::left )
        << action_id
        << "\t" << weatherData.Clouds
        << "\t" << weatherData.Precip
        << "\t" << weatherData.Vis
        << "\t" << weatherData.WindDir
        << "\t" << weatherData.WindSpeed
        << "\t" << weatherData.Sea
        << "\t" << weatherData.Temp
        << "\t" << weatherData.Barom
        << "\t" << temp<<"\t"
        << "\n";
}


WeatherDataPacket::~WeatherDataPacket(void) {
        delete (char *) action_id;
        /*
        delete (char *) Clouds;
        delete (char *) Precip;
        delete (char *) Vis;
        delete (char *) WindDir;
        delete (char *) WindSpeed;
        delete (char *) Sea;
        delete (char *) Temp;
        delete (char *) Barom;
        */
}

#ifndef COMBINE  // exclude code for combine program
```

```
// ferrypacket stuff
FerryDataCollection::FerryDataCollection(ccIndex aLimit, ccIndex aDelta) :
                TNSCollection(aLimit, aDelta) {

                fos.open (outfiles.TempFileName, ios::out); // temp outfile
                fos << FILE_MARKER_TOP << "\t"<< "\n";
}

void FerryDataCollection::writeAll(void) {
        forEach(callWrite, &fos /*file pointer here */ );
        forEach(callVisPr, (ofstream *) NULL); //see below
        VisFile.flush(); // flush out the data to the file
        if (POD.spewData==On) {
                //might want to add ios::state check here
                // would check to see if printer is still valid
                forEach(callSpew, (ofstream *) NULL);  // second argument is dummy
                PrinterStream.flush();  // flush information out
        }
        fos << flush;
}
ccIndex FerryDataCollection::insert (DataInitializationDataPacket *DIDP) {
        if (count==limit) {
                writeAll();
                freeAll();
        }
        return TNSCollection::insert(DIDP);
}
ccIndex FerryDataCollection::insert (IncidentsDataPacket *IDP) {
        if (count==limit) {
                writeAll();
                freeAll();
        }
        return TNSCollection::insert(IDP);
}

ccIndex FerryDataCollection::insert (DrillsDataPacket *DDP) {
        if (count==limit) {
                writeAll();
                freeAll();

        }
        return TNSCollection::insert(DDP);
}
ccIndex FerryDataCollection::insert (MovementDataPacket *MDP) {
        if (count==limit) {
                writeAll();
                freeAll();
        }
        return TNSCollection::insert(MDP);
}
ccIndex FerryDataCollection::insert (WeatherDataPacket *WDP) {
        if (count==limit) {
                writeAll();
                freeAll();
        }
        return TNSCollection::insert(WDP);
}
void FerryDataCollection::fileclose(void) {
        fos << FILE_MARKER_BOTTOM << "\t" << "\n";
```

```
        fos.close();
}

void callWrite(void *FDP, void *FOF /* file pointer here */) {
        // call the appropriate write function for the collection item
        ((FerryDataPacket *)FDP)->write((ofstream *) FOF);
}
void callSpew(void *FDP, void *Junk) {
        // Junk is an unnecessary call to keep consistency with call write
        ((FerryDataPacket *) FDP)->SpewPrint(&PrinterStream);
        // call the spew function for that item
}
void callVisPr(void *FDP, void *Junk) {
        // see above
        ((FerryDataPacket *) FDP )->VisPrint(&VisFile);
        // call the vis print function for that item
}
#endif  // excludes compiler offending code for the combination program
```

```
/*** this is file frrydat2.h  ****/
// this file declares data and other non gui classes for the ferrylog
// and the combine programs
//


#if!defined(__FRRYDAT2_H)
#define __FRRYDAT2_H
//#include "frrygui3.h"
#include "consts.h"
#include "pstream.h"
#define Uses_TNSCollection
#define Uses_TObject
#include <tv.h>
#include <iostream.h>
#include <time.h>
#include <fstream.h>
#include <stdio.h>  //temporary


// data structures
struct {
        char height [tides_length];
        char time[tides_length];
} typedef Tides_Struct;

struct {
        Tides_Struct high [tides_entries];
        Tides_Struct low [tides_entries];
} typedef TidesInfo;

struct {
        char crew_position_name[crew_positions][crew_name_length];
        char crew_name [crew_positions][crew_name_length];
        char watch_name[2];  // Watch A, B, C, etc.
} typedef CrewInfo;

struct {    // structure to hold information about checkpoints
             // and arrival and departure points

        int number_of_points; //number of checkpoints + arrival & departure
        // must be less than MaxCheckPoints
        char  arrival [MaxCheckLength]; //maximum name length for a field
        char  depart[MaxCheckLength];
        char  chkpoint[MaxCheckPoints-2][MaxCheckLength];
} typedef MoveInfo;

struct {
        char route_name[MaxRouteNameLength];
        char arrival [MaxRouteData_ad][MaxCheckLength];
        char depart [MaxRouteData_ad][MaxCheckLength];
        char chkpoint[MaxRouteData_ck][MaxCheckLength];
} typedef RouteInfo;


// structure to capture the current time of various actions
```

```
struct {
        time_t machineTime;
        time_t givenTime;
} typedef OccurTime;
struct {
        char vessel_name [MaxVesselNameLength];
        int vessel_end;  // 0 =one ended ferry (pass only)
                         // 1 =end one of a 2 ended ferry
                         // 2 =end two of a 2 ended ferry
                         // all others invalid

} typedef VesselInfo;
// weather data structure
class WeatherInfo {
public:
        WeatherInfo(void); // constructor
        char Clouds[MaxWeatherLength];
        char Precip[MaxWeatherLength];
        char Vis[MaxWeatherLength];
        char WindDir[MaxWeatherLength];
        char WindSpeed[MaxWeatherLength];
        char Sea[MaxWeatherLength];
        char Temp[MaxWeatherLength];
        char Barom[MaxWeatherLength];
};

/*********  declaration for data storage class  **********/
class DataStorageClass {
public:
        DataStorageClass(void);
        CrewInfo crew_data;
        TidesInfo tides_data;
        MoveInfo move_data;
        RouteInfo route_data;
        VesselInfo vessel_data;
        void data_setup(void);
        void route_setup(void);
        void move_setup(void);
/* this function replaces string elements following a null terminator
        with more null terminators */


private:
        int setup_flag;


};
class FerryDataPacket :public TObject {
public:
        virtual void write (ofstream * os)=0;
        virtual void SpewPrint (prnstream * os)=0;
        virtual void VisPrint (fstream * Vs)=0;
        FerryDataPacket(void) {}
};

class DataInitializationDataPacket: public FerryDataPacket {

public:
```

```
        CrewInfo crewInit;
        const char *action_id;  // identifies text label for this operation
        TidesInfo tidesInit;
        MoveInfo moveInit;
        RouteInfo routeInit;
        VesselInfo vesselInit;
        OccurTime initialize_time;
        // constructor
        DataInitializationDataPacket ( CrewInfo, TidesInfo, MoveInfo,
                RouteInfo, VesselInfo);
        virtual void write (ofstream *os);
        virtual void SpewPrint (prnstream *os);
        virtual void VisPrint (fstream *Vs);
        void ExcPrint (ofstream *es);
        void FinalPrint (void);
        ~DataInitializationDataPacket(void);  // destructor

};

class WeatherDataPacket:public FerryDataPacket {
public:
        WeatherInfo weatherData;
        OccurTime weatherTime;
        const char *action_id;
        WeatherDataPacket (WeatherInfo &weatherPass, OccurTime &timePass);
        virtual void write (ofstream *os);
        virtual void SpewPrint (prnstream *os);
        virtual void VisPrint (fstream *Vs);
        ~WeatherDataPacket(void);
};

class MovementDataPacket :public FerryDataPacket {

public:

        // data elements
        const char *action_id; // identifies text label for this operation
        const char *name; // name of point: seattle, duwamish head, etc
        const char *point_type;  // arrival depart, checkpoint
        OccurTime time_data; // push and edited times
        // constructor
        MovementDataPacket( char *pass_name, char *pass_type,
                OccurTime pass_time);
        virtual void write (ofstream * os);
        virtual void SpewPrint (prnstream *os);
        virtual void VisPrint (fstream *Vs);
//      virtual void read(void){}  // add to later
                                   // perhaps void * (istream&)
        ~MovementDataPacket (void); // destructor
};

class DrillsDataPacket:public FerryDataPacket {
        // data elements
public:
        const char *action_id; // identifies text label for this operation
        const char *drill_name; // name of drill
        OccurTime time_data;  // push and edited times
        const char *comment; // drill comment
```

```
        virtual void write (ofstream * os);
        virtual void SpewPrint (pmstream *os);
        virtual void VisPrint (fstream *Vs);
//      virtual void read(void) {} // same deal expand later
        // constructor
        DrillsDataPacket (char *pass_drill, OccurTime pass_time,
                char * pass_comment);
        ~DrillsDataPacket(void); //destructor
};

class IncidentsDataPacket:public FerryDataPacket {
        // data elements
public:
        const char *action_id; // identifies text label for this operation
        const char *Incident_name; // name of incident
        OccurTime time_data;  // push and edited times
        const char *comment; // incident comment
        virtual void write (ofstream * os);
        virtual void SpewPrint (pmstream *os);
        virtual void VisPrint (fstream *Vs);
//      virtual void read(void) {} // same deal expand later
        // constructor
        IncidentsDataPacket (char *pass_incident, OccurTime pass_time,
                char * pass_comment);
        ~IncidentsDataPacket(void); //destructor
};


// TNSCollection class
class FerryDataCollection: public TNSCollection {
public:
        FerryDataCollection(ccIndex aLimit, ccIndex aDelta) ;

        ccIndex insert (MovementDataPacket *MDP);
        ccIndex insert (DrillsDataPacket *DDP);
        ccIndex insert (IncidentsDataPacket *IDP);
        ccIndex insert (DataInitializationDataPacket *DIDP);
        ccIndex insert (WeatherDataPacket *WDP);
        void writeAll (void);
        ofstream fos;
        void fileclose(void);
};
void callWrite( void *FDP, void *FOF );  // has to not be a class member function
void callSpew(void *FDP, void *FOF);  // second argument is dummy
void callVisPr(void *FDP, void *FOF);  // second argument is dummy
// FerryDataPacket is FDP type
//FOF is Ferry Output File  and is a pointer to an ofstream


// data structures

// structure to contain file names for the data

struct {
        char FinalFileName[13];
        char TempFileName[13];
} typedef FerryFileNames;

#endif  //__FRRYDAT2_H
```

FRRYGUI3.H

```
/************ This is a header file for the graphical ********************
********************** interface section of the vessel log *********/

/* this file is frrygui3.h */


// the main objectives of this version are to provide good encapsulation
//   of the various windows classes
// this file contains class declarations for various graphical elements

#if!defined(__FRRYGUI3_H)
#define __FRRYGUI3_H


//  define uses section
//#define Uses_TView
//#define Uses_TWindow
//#define Uses_TEvent
//#define Uses_TDialog
//#define Uses_TWindowInit
//#define Uses_TButton
//#define Uses_TInputLine


// include files
#include <tv.h>
#include <time.h>
#include "myvis2.h"
#include "consts.h"
#include "frrydat2.h"
#include "utilfncs.h"
#include "tmdis.h"
// this constant is here because maxViewWidth is defined in the TVision
// library
const int maxLineLength =maxViewWidth+1;


/*****   Tides viewer Class  *******/

class TidesViewer : public TWindow {    // uses TidesInterior to draw the
                                        // interior view
public:
        TidesViewer (const TRect &r, short aNumber,
                TidesInfo tides_pass); // constructor
        virtual void handleEvent (TEvent & event);  // override standard

};
class TidesInterior: public TView {

public:
        TidesInterior (const TRect &bounds, TidesInfo &tides_pass); // constructor
        virtual void draw(); //override TView::draw
private:
        TidesInfo tides_data;
        int print(int *cur_line_ptr, char first[]=" ",
                        char second[]=" ");
```

```
};

/*****        Crew Viewer Classes ****/
class CrewViewer: public TWindow {

public:
        CrewViewer ( const TRect &bounds, short aNumber,
                CrewInfo &crew_pass );

        virtual void handleEvent (TEvent & event);

};
class CrewViewerInterior: public TView {

public:
        CrewViewerInterior (const TRect & bounds,  CrewInfo &crew_pass);
        virtual void draw (void);
        int print(int *cur_line_ptr, char first[]=" ");

private:
        CrewInfo crew_data;
        char * strsetup( char *outstr, char * instr1,char *instr2);
};
/****  movement Viewer class   **************/

class MovementViewer: public PrimaryDialog {
private:
        MoveInfo chkpoint_data;
public:
        MovementViewer (MoveInfo chk_data);
        virtual void handleEvent (TEvent &event);
        TRect ButtonSizer(int startbutton, int lwidth, int number);
        void myChkpointConfirm(int);
        void myArrivalConfirm(void);
        void myDepartConfirm(void);
};

/*** classes used by movement Viewer  ******/

/* time confirm modal dialog  */
class MvTimeConfirm: public SecondaryDialog {
private:
        OccurTime time_info;
        char GeoPoint[MaxCheckLength];
        void data_output(void);
        char point_type[MaxCommandLength];
public:
        MvTimeConfirm(char Cpoint[MaxCheckLength],
                const char  pass_type[MaxCommandLength]);
                        // pass type is arrive,depart,chkpnt
        virtual void handleEvent (TEvent &);
};

class TimeGrabber: public SecondaryDialog {
// grabs given time from user and checks it
private:
        char inputtime[6];
        TimeDisplay TGTime;
```

```
        int checkData(void);
        int *status_ptr;
        virtual void draw(void);


public:
        OccurTime *time_ptr;  // needs to be public cause we pass it to
                              //      another class
        TimeGrabber(OccurTime *, int *);
        virtual void handleEvent(TEvent &event);
};

class BadTimeDialog: public SecondaryDialog {
public:
        BadTimeDialog();
};

class WeirdTimeDialog: public SecondaryDialog {
public:
        WeirdTimeDialog(int *);
        virtual void handleEvent(TEvent &);

private:
        int *w_ptr;
};

/* Drills Viewer and related classes */
class DrillsViewer: public PrimaryDialog {

public:
        DrillsViewer ();
        virtual void handleEvent (TEvent &event);
        void myDrillConfirm(const char *);

};

class DrTimeConfirm: public SecondaryDialog {
private:
        OccurTime time_info;
        char DrillName[MaxDrillNameLength];
        void data_output(void);

public:
        DrTimeConfirm(const char *Dname);
        virtual void handleEvent (TEvent &);
        char Our_comment[MaxCommentLength];
};

class VesselInit: public PrimaryDialog {
public:
        VesselInit(VesselInfo *);
        virtual void handleEvent (TEvent &);
        VesselInfo *vessel_data_ptr;

};
class VesselEndInit: public PrimaryDialog {
public:
        VesselEndInit(VesselInfo *);
        virtual void handleEvent (TEvent &);
```

```
        VesselInfo *vessel_data_ptr;

};

class WatchInit: public PrimaryDialog {
public:
        WatchInit(CrewInfo *);
        virtual void handleEvent (TEvent &);
        CrewInfo *watch_data_ptr;
};

class RouteInit: public PrimaryDialog {
public:
        RouteInit(RouteInfo *route_data_pass);
        virtual void handleEvent (TEvent &);
        RouteInfo *route_data_ptr;

};
class ArrivalInit:public PrimaryDialog {
public:
        ArrivalInit (MoveInfo *, RouteInfo *);
        virtual void handleEvent (TEvent &);
        MoveInfo *selected_data_ptr;
        RouteInfo *possible_data_ptr;
private:
        TRect ButtonSizer(int startbutton, int lwidth, int number);
        int non_empty_fields(void);
};
class DepartureInit:public PrimaryDialog {
public:
        DepartureInit (MoveInfo *, RouteInfo *);
        virtual void handleEvent (TEvent &);
        MoveInfo *selected_data_ptr;
        RouteInfo *possible_data_ptr;
private:
        TRect ButtonSizer(int startbutton, int lwidth, int number);
        int non_empty_fields(void);
};

class CrewInit: public PrimaryDialog {
public:
        CrewInit(CrewInfo *crew_data_pass);
        virtual void handleEvent (TEvent &);
        CrewInfo *crew_data_ptr;

};
class TidesInit: public PrimaryDialog {
public:
        TidesInit(TidesInfo *tides_data_pass);
        virtual void handleEvent (TEvent &);
        TidesInfo *tides_data_ptr;
};



// template version of commentGrabber
template <class Type>
class CommentGrabber: public SecondaryDialog {
```

```
public:
        CommentGrabber(Type *);
        virtual void handleEvent (TEvent &);
        char comment_ptr[MaxCommentLength];
        Type *class_ptr;

};

// end template version
/**********    incidents viewer    *******/

class IncidentsViewer: public PrimaryDialog {
public:
        IncidentsViewer ();
        virtual void handleEvent (TEvent &event);
        void myIncidentsConfirm(const char *incidentmsg);
};

class InTimeConfirm: public SecondaryDialog {
private:
        OccurTime time_info;
        char IncidentName[MaxInciNameLength];
        void data_output(void);

public:
        InTimeConfirm(const char * Iname);
        //      virtual TPalette &getPalette() const;
        virtual void handleEvent (TEvent &);
        char Our_comment[MaxCommentLength];
};

// class intended for main scope
// store data from initialization and pass to gui


#endif  //__FRRYGUI3_H
```

FRRYLOG.BAT

```
echo off
cls
echo Please enter the current time.  If correct, press enter
time
echo Please enter the current date.  If correct, press enter
date
del visual.dat
ferrylog.exe
```

```
/*-------------------------------------------------------*/
/*                                                       */
/*   Turbo Vision 1.0                                    */
/*   Turbo Vision Demo                                   */
/*   Copyright (c) 1991 by Borland International         */
/*                                                       */
/*   Gadgets.cpp: Gadgets for the Turbo Vision Demo. Includes a   */
/*       heap view and a clock view which display the clock at the */
/*       right end of the menu bar and the current heap space at   */
/*      the right end of the status line.                */
/*-------------------------------------------------------*/

// This file was adapted by Erik beck for use in the ferrylog program
// and was renamed gad.cpp
// this file contains a heap viewer and a clock viewer that show up on
/* the user screen.  At some point the heap viewer should be removed
from the users view. */


#define Uses_TRect
#define Uses_TView
#define Uses_TDrawBuffer
#include <tv.h>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <alloc.h>
#include <ctype.h>
#include <strstrea.h>
#include <iomanip.h>

#include "gadgets.h"


//
// ------------ Clock Viewer functions
//

TClockView::TClockView( TRect& r ) : TView( r )
{
   strncpy(lastTime, " ",26);   // string length of 9
   strncpy(curTime,  " ",26);

}


void TClockView::draw()
{
   TDrawBuffer buf;
   char c = getColor(2);

   buf.moveChar(0, ' ', c, size.x);
   buf.moveStr(0,curTime,c);
   writeLine(0, 0, size.x, 1, buf);
```

```
}

void TClockView::update()
{
   time_t t = time(NULL);
   char date[26];
   strncpy(date, ctime(&t),26);
   date[24]='\0' ;  //eliminate the /n cause it looks funny

   strcpy(curTime, date);      /* Extract time. */


    if( strcmp(lastTime, curTime) )
          {
           drawView();
          strncpy(lastTime, curTime,26);
          }
}


//
// ------------ Heap Viewer functions
//

THeapView::THeapView(TRect& r) : TView( r )
{
   oldMem = 0;
   newMem = heapSize();
}


void THeapView::draw()
{
   TDrawBuffer buf;
   char c = getColor(2);

   buf.moveChar(0, ' ', c, size.x);
   buf.moveStr(0, heapStr, c);
   writeLine(0, 0, size.x, 1, buf);
}


void THeapView::update()
{
   if( (newMem = heapSize()) != oldMem )
     {
      oldMem = newMem;
      drawView();
     }
}


long THeapView::heapSize()
{
   long total = farcoreleft();
   struct farheapinfo heap;
```

```
    ostrstream totalStr( heapStr, sizeof heapStr);

    switch( farheapcheck() )
     {
      case _HEAPEMPTY:
         strcpy(heapStr, "    No heap");
        total = -1;
        break;

      case _HEAPCORRUPT:
         strcpy(heapStr, "Heap corrupt");
        total = -2;
        break;

      case _HEAPOK:
         heap.ptr = NULL;
         while(farheapwalk(&heap) != _HEAPEND)
           if(!heap.in_use)
              total += heap.size;
         totalStr << setw(12) << total << ends;
        break;
     }
    return(total);
}
```

GADGETS.H

```
/*------------------------------------------------*/
/*                                                */
/*  Turbo Vision 1.0                              */
/*  Copyright (c) 1991 by Borland International   */
/*                                                */
/*  Gadgets.h : Header file for gadgets.cpp       */
/*------------------------------------------------*/
// this file is gadgets.h and was modified by erik Beck
// This file is used in the ferrylog program and contains class
// declarations for the clock and heap viewers


#if !defined( __GADGETS_H )
#define __GADGETS_H

#define Uses_TEvent
#define Uses_TRect
#define Uses_TView
#include <tv.h>


class TClockView : public TView
{

public:

   TClockView( TRect& r );
   virtual void draw();
   virtual void update();

private:

   char lastTime[26];
   char curTime[26];

};


class THeapView : public TView
{

public:

   THeapView( TRect& r );
   virtual void update();
   virtual void draw();
   virtual long heapSize();

private:

   long oldMem, newMem;
   char heapStr[16];

};

#endif    // __GADGETS_H
```

GUIDRI2.CPP

```
// This file is guidri2.cpp and is used in the ferrylog program

// class definition file for the tidesviever, crewviewer, and the movement
// subportions of the WSF Gui project
// graphical elements


//#define Uses_TDrawBuffer
//#define Uses_TWindow
#define Uses_TEvent
#define Uses_TView
//#define Uses_TApplication
//#define Uses_TStatusLine
//#define Uses_TMenuBar
//#define Uses_TDeskTop
//#define Uses_TStatusDef
//#define Uses_TStatusItem
//#define Uses_TSubMenu
//#define Uses_TMenuItem
#define Uses_TKeys
//#define Uses_TProgram
//#define Uses_TInputLine
#define Uses_TLabel
#define Uses_TButton

// include files

#include "frrydat2.h"
#include "frrygui3.h"
#include <tv.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>
// #include <dos.h>  // for sound


// test global
extern FerryDataCollection  FerryData_global;



/***    Drills Viewer  *****/

DrillsViewer::DrillsViewer(void):
        PrimaryDialog( TRect(10,0,45,19), "Drills"), TWindowInit (initFrame) {
        // copy data elements

        TButton *AB_button=new TButton (TRect (2,2,20,5),
                "Abandon Ship", cmdrillAbS, bfNormal);
        insert (AB_button);

        TButton *RB_button =new TButton (TRect(2,6,20,9),
                "Rescue Boat", cmdrillRB, bfNormal);
        insert (RB_button);
```

```
        TButton *F_button =new TButton (TRect(2,10,20,13),
                "Fire Drill", cmdrillF, bfNormal);
        insert (F_button);

        AB_button->select();


}


void DrillsViewer::handleEvent (TEvent &event) {

        PrimaryDialog::handleEvent(event);
        if (event.what==evBroadcast) {
                switch (event.message.command) {
                        case msgDrillsViewer:
                                clearEvent(event);
                        break;
                }
        }
        if (event.what==evCommand) {       /* this code determines which
                                checkpoint button was pushed */
                switch (event.message.command) {
                        case cmdrillAbS:
                                clearEvent(event);
                                myDrillConfirm(ABANDON_TXT);
                        break;
                        case cmdrillRB:
                                clearEvent(event);
                                myDrillConfirm(RESCUE_TXT);
                        break;
                        case cmdrillF:
                                clearEvent(event);
                                myDrillConfirm(FIRE_TXT);
                        break;
                }

        }
}
void DrillsViewer::myDrillConfirm(const char *drillmsg) {

        DrTimeConfirm *timefirm=new
                        DrTimeConfirm(drillmsg);
                this->execView (validView(timefirm));
         destroy (timefirm);
}

/* Drill time confirm modal dialog  */

DrTimeConfirm::DrTimeConfirm(const char *Dname):
        SecondaryDialog (TRect (2,2,35,17),"How Long Ago?"),
        TWindowInit (initFrame) {

        // initialize occur time
        time_info.machineTime=0L;
        time_info.givenTime=0L;
        // initialize Drill Type

        strncpy(DrillName,Dname,MaxDrillNameLength);
```

```
        // initialize Comment data
        strncpy(Our_comment," ", MaxCommentLength);

        insert (new TLabel (TRect(2,2,30,3), "When did we have our drill?",0));
        insert (new TLabel (TRect(2,4,17,5), DrillName, 0));

        TButton *JN =new TButton (TRect(2,7,15,9), "Just Now", cmOccurNow,
                bfDefault);
        insert (JN);
        insert (new TButton (TRect(17,7,28,9), "Earlier", cmOccurBefore,
                bfNormal));
        insert (new TButton (TRect(2,10,25,12),
                "OOps! Go Back", cmCancel, bfNormal));

        JN->select();

}
void DrTimeConfirm::handleEvent(TEvent &event) {
        // time_t time_out;
        if (event.what!=evCommand) SecondaryDialog::handleEvent(event);
        if (event.what==evCommand && event.message.command==cmCancel)
                SecondaryDialog::handleEvent(event);

        if (event.what==evCommand && event.message.command!=cmCancel) {
                switch(event.message.command) {
                        case cmOccurNow:
                                clearEvent(event);

                                // get the time of the button press
                                // time_out=time (NULL);
                                time_info.givenTime=
                                        time_info.machineTime=
                                        time(NULL);

                                // get a comment
                                CommentGrabber<DrTimeConfirm>
*comment_window=
                                new CommentGrabber<DrTimeConfirm>
                                 (this);
                                this ->execView (validView(comment_window));
                        destroy (comment_window);

                                // facility for closing up the window
                                event.what = evCommand;
                                event.message.command = cmClose;
                                putEvent( event );


                        break;
                        case cmOccurBefore:
                                clearEvent(event);
                                // get the time of the button press
                          //    time_out=time (NULL);
                                time_info.givenTime=
                                        time_info.machineTime=
                                        time(NULL);

                                int status=0;
                                // checks the status of the
```

```
                                        // time grabber window
                                        // valid if altered,
                                        // invalid if oops is pressed


                                        TimeGrabber *timeWindow =
                                                // below arg changed from pointer
                                                new TimeGrabber (&time_info, &status);
                                        this ->execView(validView(timeWindow));
                        destroy (timeWindow);
                                        // Now close this window if data was valid
                                        if (status==VALID) {
                                                CommentGrabber<DrTimeConfirm>
                                                *comment_window=
                                                  new CommentGrabber<DrTimeConfirm>
                                                        (this);
                                                        //note template

                                                this ->execView
                                                        (validView(comment_window));
                                                destroy (comment_window);
                                                data_output();
                                                // now close the window
                                                event.what = evCommand;
                                                event.message.command = cmCancel;
                                                putEvent( event );
                                        }

                                break;
                                case cmClose:
                                        clearEvent(event);
                                        data_output();
                                        // now close the window
                                        event.what = evCommand;
                                        event.message.command = cmCancel;
                                        putEvent( event );
                                break;

                        }
                }
}
void DrTimeConfirm::data_output(void) {

        MakeASound(sound_record);
        DrillsDataPacket * DDP =new DrillsDataPacket(DrillName,time_info,
                Our_comment);
        assert (DDP!=NULL);
        FerryData_global.insert(DDP);
}


// template version of CommentGrabber constructor
template <class Type>

CommentGrabber<Type>::CommentGrabber(Type *class_pass):
        SecondaryDialog (TRect (0,0,30,12),"Comment"),
        TWindowInit (initFrame) {

        class_ptr=class_pass;
```

```
        // initalize the member function pointer
        strncpy (comment_ptr,"",MaxCommentLength);
        insert (new TLabel (TRect(4,1,25,2),
                "Enter your comment",0) );
        insert (new TInputLine (TRect (2,4,28,5),MaxCommentLength));
        insert (new TButton (TRect(2,7,10,10), "OK", cmOK, bfNormal));
        /*
        insert (new TButton (TRect(2,12,25,14),
                "OOps!  Go Back", cmCancel, bfNormal));
        */
        selectNext(False);
        setData(comment_ptr);

}
// end template version

template <class Type>
void CommentGrabber<Type>::handleEvent (TEvent &event) {

        if (event.what!=evCommand) SecondaryDialog::handleEvent(event);
        if (event.what==evCommand && (event.message.command!=cmNo &&
                event.message.command!=cmOK))
                SecondaryDialog::handleEvent(event);


        if (event.what==evCommand && event.message.command==cmOK) {

                getData(comment_ptr);
                strncpy(class_ptr->Our_comment, comment_ptr,
                        MaxCommentLength);
                clearEvent(event);

                // close itself
                event.what = evCommand;
                event.message.command = cmCancel;
                putEvent( event );

        }

}

/*******   incident classes  **************/


/***    Incidents Viewer  *****/

IncidentsViewer::IncidentsViewer(void):
        PrimaryDialog( TRect(1,1,64,21), "Incidents"), TWindowInit (initFrame) {
        // copy data elements


        TButton *SLIP_button=new TButton (TRect (2,2,25,4),
                "Wait for Slip", cmincidentSLIP, bfNormal);
        insert (SLIP_button);

        TButton *VES_button =new TButton (TRect(26,2,52,4),
                "Heavy Vessel Traffic", cmincidentVESSEL, bfNormal);
        insert (VES_button);
```

```
        TButton *AUTO_button =new TButton (TRect(2,5,25,7),
                "Heavy Auto Traffic", cmincidentAUTO, bfNormal);
        insert (AUTO_button);

        TButton *CREW_button =new TButton (TRect(26,5,51,7),
                "Crewing Problem", cmincidentCREW, bfNormal);
        insert (CREW_button);

        TButton *PASS_button =new TButton (TRect(2,8,25,10),
                "Passenger Problem", cmincidentPASS, bfNormal);
        insert (PASS_button);

        TButton *CG_button =new TButton (TRect(26,8,51,10),
                "Coast Guard Request", cmincidentCG, bfNormal);
        insert (CG_button);

        TButton *WEA_button =new TButton (TRect(2,11,25,13),
                "Weather or Tides", cmincidentWEATHER, bfNormal);
        insert (WEA_button);

        TButton *MEC_button =new TButton (TRect(26,11,51,13),
                "Mechanical", cmincidentMECHAN, bfNormal);
        insert (MEC_button);

        TButton *POL_button =new TButton (TRect(2,14,25,16),
                "Police, aid car, etc", cmincidentPOLICE, bfNormal);
        insert (POL_button);

        TButton *OTH_button =new TButton (TRect (26,14,51,16),
                "Other incidents", cmincidentOTHER, bfNormal);
        insert (OTH_button);

        SLIP_button ->select(); // make the first button the default


}


void IncidentsViewer::handleEvent (TEvent &event) {

        PrimaryDialog::handleEvent(event);
        if (event.what==evBroadcast) {
                switch (event.message.command) {
                        case msgIncidentsViewer:
                                clearEvent(event);
                        break;
                }
        }
        if (event.what==evCommand) {       /* this code determines which
                                checkpoint button was pushed */
                switch (event.message.command) {
                        case cmincidentSLIP:
                                clearEvent(event);
                                myIncidentsConfirm(INCI_SLIP);
                        break;
                        case cmincidentVESSEL:
                                clearEvent(event);
```

```
                        myIncidentsConfirm(INCI_VESSEL);
                    break;
                    case cmincidentAUTO:
                        clearEvent(event);
                        myIncidentsConfirm(INCI_AUTO);
                    break;
                    case cmincidentCREW:
                        clearEvent(event);
                        myIncidentsConfirm(INCI_CREW);
                    break;
                    case cmincidentPASS:
                        clearEvent(event);
                        myIncidentsConfirm(INCI_PASS);
                    break;
                    case cmincidentWEATHER:
                        clearEvent(event);
                        myIncidentsConfirm(INCI_WEATHER);
                    break;
                    case cmincidentMECHAN:
                        clearEvent(event);
                        myIncidentsConfirm(INCI_MECHAN);
                    break;
                    case cmincidentPOLICE:
                        clearEvent(event);
                        myIncidentsConfirm(INCI_POLICE);
                    break;
                    case cmincidentCG:
                        clearEvent(event);
                        myIncidentsConfirm(INCI_COASTGU);
                    break;
                    case cmincidentOTHER:
                        clearEvent(event);
                        myIncidentsConfirm(INCI_OTHER);
                    break;

                }

        }
}
void IncidentsViewer::myIncidentsConfirm(const char *incidentmsg) {

        InTimeConfirm *timefirm=new
                        InTimeConfirm(incidentmsg);
                this->execView(timefirm);
        destroy(timefirm);
}

/* incident  time confirm modal dialog  */

InTimeConfirm::InTimeConfirm(const char *lname):
        SecondaryDialog (TRect (2,2,45,15),"How Long Ago?"),
        TWindowInit (initFrame) {


        // initialize occur time
        time_info.machineTime=0L;
        time_info.givenTime=0L;
        // initialize Incident Type
```

```
        strncpy(IncidentName,Iname,MaxInciNameLength);
        // initialize Comment data
        strncpy(Our_comment," ", MaxCommentLength);

        insert (new TLabel (TRect(2,2,41,3), "When did this incident happen?"
                ,0));
        insert (new TLabel (TRect(2,4,30,5), IncidentName, 0));

        TButton *JN=new TButton (TRect(2,7,15,9), "Just Now", cmOccurNow,
                bfDefault);
        insert (JN);
        insert (new TButton (TRect(17,7,28,9), "Earlier", cmOccurBefore,
                bfNormal));
        insert (new TButton (TRect(2,10,25,12),
                "OOps! Go Back", cmCancel, bfNormal));

        JN->select();

}
void InTimeConfirm::handleEvent(TEvent &event) {
        // time_t time_out;
        if (event.what!=evCommand) SecondaryDialog::handleEvent(event);
        if (event.what==evCommand && event.message.command==cmCancel)
                SecondaryDialog::handleEvent(event);

        if (event.what==evCommand && event.message.command!=cmCancel) {
                switch(event.message.command) {
                        case cmOccurNow:
                                clearEvent(event);

                                // get the time of the button press
                                // time_out=time (NULL);
                                time_info.givenTime=
                                        time_info.machineTime=
                                        time(NULL);

                                // get a comment
                                CommentGrabber<InTimeConfirm> *comment_window=
                                new CommentGrabber<InTimeConfirm>
                                 (this);
                                this ->execView (validView(comment_window));
                                destroy (comment_window);
                                // facility for closing up the window
                                event.what = evCommand;
                                event.message.command = cmClose;
                                putEvent( event );


                        break;
                        case cmOccurBefore:
                                clearEvent(event);
                                // get the time of the button press
                           //   time_out=time (NULL);
                                time_info.givenTime=
                                        time_info.machineTime=
                                        time(NULL);

                                int status=0;
                                // checks the status of the
```

```
                                        // time grabber window
                                        // valid if altered,
                                        // invalid if oops is pressed


                                        TimeGrabber *timeWindow =
                                                // below arg changed from pointer
                                                new TimeGrabber (&time_info, &status);
                                        this ->execView(validView(timeWindow));
                                        destroy (timeWindow);
                                        // Now close this window if data was valid
                                        if (status==VALID) {
                                                CommentGrabber<InTimeConfirm>
                                                *comment_window=
                                                 new CommentGrabber<InTimeConfirm>
                                                        (this);
                                                        //note template

                                                this ->execView
                                                        (validView(comment_window));
                                                destroy (comment_window);
                                                data_output();
                                                // now close the window
                                                event.what = evCommand;
                                                event.message.command = cmCancel;
                                                putEvent( event );
                                        }
                                break;
                                case cmClose:
                                        clearEvent(event);
                                        data_output();
                                        // now close the window
                                        event.what = evCommand;
                                        event.message.command = cmCancel;
                                        putEvent( event );
                                break;

                        }
                }
        }
void InTimeConfirm::data_output(void) {

        MakeASound(sound_record);
        IncidentsDataPacket *IDP = new IncidentsDataPacket
                (IncidentName,time_info, Our_comment );
        assert (IDP!=NULL);
        FerryData_global.insert(IDP);


}
```

```
/*  this is file guiini2.cpp  */
/* it contains the declarations for gui elements in the initializer */

// the classes here are used to enter the initial info about
// crew, watch, vessel, route, etc.
#define Uses_Twindow
#define Uses_TLabel
#define Uses_TButton
#define Uses_TEvent

#include "frrydat2.h"
#include "frrygui3.h"
#include <tv.h>
#include <string.h>

void WatchCrewLoad(CrewInfo *crew_ptr);
void WatchCrewSave(CrewInfo *crew_ptr);

// vessel init view  initialize name of the vessel
VesselInit::VesselInit (VesselInfo *vessel_data_pass):
        PrimaryDialog (TRect (0,0,79,22), "Vessel Identification"),
        TWindowInit (initFrame) {
        /* put the body of the view here */
        /* turn little box off */
        flags ^=wfClose;
        vessel_data_ptr=vessel_data_pass;
        insert (new TLabel (TRect (2,2,25,3),
                "Identify your vessel!",0) );
        TButton * default_button= new TButton (TRect(1,4,15,6),
                VES_NAME_CATH,
                id_ves_cath, bfNormal);
        insert (default_button);


        insert (new TButton (TRect(16,4,30,6), VES_NAME_CHEL,
                id_ves_chel, bfNormal));
        insert (new TButton (TRect(31,4,45,6), VES_NAME_ELWH,
                id_ves_elwh, bfNormal));
        insert (new TButton (TRect(46,4,60,6), VES_NAME_EVEG,
                id_ves_eveg, bfNormal));
        insert (new TButton (TRect(61,4,75,6), VES_NAME_HIYU,
                id_ves_hiyu, bfNormal));

        insert (new TButton (TRect(1,6,15,8), VES_NAME_HYAK,
                id_ves_hyak, bfNormal));
        insert (new TButton (TRect(16,6,30,8), VES_NAME_ILLA,
                id_ves_illa, bfNormal));
        insert (new TButton (TRect(31,6,45,8), VES_NAME_ISSA,
                id_ves_issa, bfNormal));
        insert (new TButton (TRect(46,6,60,8), VES_NAME_KALA,
                id_ves_kala, bfNormal));
        insert (new TButton (TRect(61,6,75,8), VES_NAME_KALE,
                id_ves_kale, bfNormal));
        insert (new TButton (TRect(1,8,15,10), VES_NAME_KITS,
                id_ves_kits, bfNormal));
        insert (new TButton (TRect(16,8,30,10), VES_NAME_KITT,
```

```
                id_ves_kitt, bfNormal));
        insert (new TButton (TRect(31,8,45,10), VES_NAME_KLAH,
                id_ves_klah, bfNormal));
        insert (new TButton (TRect(46,8,60,10), VES_NAME_KLIC,
                id_ves_klic, bfNormal));
        insert (new TButton (TRect(61,8,75,10), VES_NAME_NISQ,
                id_ves_nisq, bfNormal));
        insert (new TButton (TRect(1,10,15,12), VES_NAME_OLYM,
                id_ves_olym, bfNormal));
        insert (new TButton (TRect(16,10,30,12), VES_NAME_QUIN,
                id_ves_quin, bfNormal));
        insert (new TButton (TRect(31,10,45,12), VES_NAME_RHOD,
                id_ves_rhod, bfNormal));
        insert (new TButton (TRect(46,10,60,12), VES_NAME_SEAL,
                id_ves_seal, bfNormal));
        insert (new TButton (TRect(61,10,75,12), VES_NAME_SKAG,
                id_ves_skag, bfNormal));
        insert (new TButton (TRect(1,12,15,14), VES_NAME_SPOK,
                id_ves_spok, bfNormal));
        insert (new TButton (TRect(16,12,30,14), VES_NAME_TILL,
                id_ves_till, bfNormal));
        insert (new TButton (TRect(31,12,45,14), VES_NAME_TYEE,
                id_ves_tyee, bfNormal));
        insert (new TButton (TRect(46,12,60,14), VES_NAME_WALL,
                id_ves_wall, bfNormal));
        insert (new TButton (TRect(61,12,75,14), VES_NAME_YAKI,
                id_ves_yaki, bfNormal));
        default_button->select();

}

void VesselInit::handleEvent (TEvent & event) {
        PrimaryDialog::handleEvent (event);

        if (event.what==evCommand) { /* this code records which
                                        vessel button was pressed */

                if(event.message.command >=id_ves_start &&
                        event.message.command <=id_ves_end) {
                        strncpy (vessel_data_ptr->vessel_name,
                                VesselNameLookup(event.message.command),
                                MaxVesselNameLength);

                        clearEvent(event);
                        event.what=evCommand;
                        event.message.command=cmCancel;
                        putEvent(event);
                }
        }
}


// vessel init view  initialize name of the vessel
VesselEndInit::VesselEndInit (VesselInfo *vessel_data_pass):
        PrimaryDialog (TRect (0,0,79,22), "Vessel End Identification"),
        TWindowInit (initFrame) {
        /* put the body of the view here */
        /* turn little box off */
        flags ^=wfClose;
```

```
        vessel_data_ptr=vessel_data_pass;
        insert (new TLabel (TRect (2,2,45,3),
                "Identify the current vessel end !",0) );
        TButton * end_one_button= new TButton (TRect(1,4,15,6),
                "End # One",
                id_ves_end_one, bfNormal);
        insert (end_one_button);

        TButton * end_two_button = new TButton (TRect (1,7,15,9),
                "End # Two",
                id_ves_end_two, bfNormal);
        insert (end_two_button);
        TButton *end_only_button = new TButton (TRect (1,10,35,12),
                "Only End (passenger only)",
                id_ves_end_only, bfNormal);
        insert (end_only_button);
        // make default
        end_one_button ->select();
}

void VesselEndInit::handleEvent (TEvent & event) {
        PrimaryDialog::handleEvent (event);

        if (event.what==evCommand) { /* this code records which
                                        vessel button was pressed */

                switch (event.message.command) {
                        case id_ves_end_only:
                                clearEvent(event);
                                vessel_data_ptr->vessel_end=0;

                                // dialog close
                                event.what=evCommand;
                                event.message.command=cmCancel;
                                putEvent(event);


                        break;

                        case id_ves_end_one:
                                clearEvent(event);
                                vessel_data_ptr->vessel_end=1;

                                // dialog close
                                event.what=evCommand;
                                event.message.command=cmCancel;
                                putEvent(event);

                        break;

                        case id_ves_end_two:
                                clearEvent(event);
                                vessel_data_ptr->vessel_end=2;

                                // dialog close
                                event.what=evCommand;
                                event.message.command=cmCancel;
                                putEvent(event);
                        break;
```

```
                }
        }
}

// ***** Watch init view

// vessel init view  initialize name of the vessel
WatchInit::WatchInit (CrewInfo *watch_data_pass):
        PrimaryDialog (TRect (0,0,79,22), "Watch Identification"),
        TWindowInit (initFrame) {
        /* put the body of the view here */
        /* turn little box off */
        flags ^=wfClose;
        watch_data_ptr=watch_data_pass;
        insert (new TLabel (TRect (2,2,45,3),
                "Identify your Watch !",0) );
        TButton * A_button= new TButton (TRect(1,4,15,6),
                "Watch A",
                id_watch_A, bfNormal);
        insert (A_button);

        TButton * B_button = new TButton (TRect (1,7,15,9),
                "Watch B",
                id_watch_B, bfNormal);
        insert (B_button);
        TButton *C_button = new TButton (TRect (1,10,15,12),
                "Watch C",
                id_watch_C, bfNormal);
        insert (C_button);

        TButton *D_button = new TButton (TRect (1,13,15,15),
                "Watch D",
                id_watch_D, bfNormal);
        insert (D_button);

        TButton *E_button = new TButton (TRect (1,16,15,18),
                "Watch E",
                id_watch_E, bfNormal);
        insert (E_button);

        TButton *F_button = new TButton (TRect (16,4,30,6),
                "Watch F",
                id_watch_F, bfNormal);
        insert (F_button);

        TButton *G_button = new TButton (TRect (16,7,30,9),
                "Watch G",
                id_watch_G, bfNormal);
        insert (G_button);

        TButton *H_button = new TButton (TRect (16,10,30,12),
                "Watch H",
                id_watch_H, bfNormal);
        insert (H_button);

        TButton *I_button = new TButton (TRect (16,13,30,15),
```

```
                "Watch I",
                id_watch_I, bfNormal);
        insert (I_button);

        TButton *J_button = new TButton (TRect (16,16,30,18),
                "Watch J",
                id_watch_J, bfNormal);
        insert (J_button);

        TButton *K_button = new TButton (TRect (31,4,45,6),
                "Watch K",
                id_watch_K, bfNormal);
        insert (K_button);

        // make default
        A_button ->select();
}

void WatchInit::handleEvent (TEvent & event) {
        PrimaryDialog::handleEvent (event);

        if (event.what==evCommand) { /* this code records which
                                        vessel button was pressed */

                switch (event.message.command) {
                        case id_watch_A:
                                clearEvent(event);
                                strcpy(watch_data_ptr->watch_name,ID_WATCH_A);

                                // dialog close
                                event.what=evCommand;
                                event.message.command=cmCancel;
                                putEvent(event);


                        break;

                        case id_watch_B:
                                clearEvent(event);
                                strcpy(watch_data_ptr->watch_name,ID_WATCH_B);

                                // dialog close
                                event.what=evCommand;
                                event.message.command=cmCancel;
                                putEvent(event);

                        break;

                        case id_watch_C:
                                clearEvent(event);
                                strcpy(watch_data_ptr->watch_name,ID_WATCH_C);

                                // dialog close
                                event.what=evCommand;
                                event.message.command=cmCancel;
                                putEvent(event);
                        break;
                        case id_watch_D:
                                clearEvent(event);
```

```
        strcpy(watch_data_ptr->watch_name,ID_WATCH_D);

        // dialog close
        event.what=evCommand;
        event.message.command=cmCancel;
        putEvent(event);
break;

case id_watch_E:
        clearEvent(event);
        strcpy(watch_data_ptr->watch_name,ID_WATCH_E);

        // dialog close
        event.what=evCommand;
        event.message.command=cmCancel;
        putEvent(event);
break;
case id_watch_F:
        clearEvent(event);
        strcpy(watch_data_ptr->watch_name,ID_WATCH_F);

        // dialog close
        event.what=evCommand;
        event.message.command=cmCancel;
        putEvent(event);
break;
case id_watch_G:
        clearEvent(event);
        strcpy(watch_data_ptr->watch_name,ID_WATCH_G);

        // dialog close
        event.what=evCommand;
        event.message.command=cmCancel;
        putEvent(event);
break;

case id_watch_H:
        clearEvent(event);
        strcpy(watch_data_ptr->watch_name,ID_WATCH_H);

        // dialog close
        event.what=evCommand;
        event.message.command=cmCancel;
        putEvent(event);
break;

case id_watch_I:
        clearEvent(event);
        strcpy(watch_data_ptr->watch_name,ID_WATCH_I);

        // dialog close
        event.what=evCommand;
        event.message.command=cmCancel;
        putEvent(event);
break;
case id_watch_J:
        clearEvent(event);
        strcpy(watch_data_ptr->watch_name,ID_WATCH_J);
```

```
                        // dialog close
                        event.what=evCommand;
                        event.message.command=cmCancel;
                        putEvent(event);
                break;
                case id_watch_K:
                        clearEvent(event);
                        strcpy(watch_data_ptr->watch_name,ID_WATCH_K);

                        // dialog close
                        event.what=evCommand;
                        event.message.command=cmCancel;
                        putEvent(event);
                break;

        }
    }
}


/**** route class declaration  ****/

RouteInit::RouteInit (RouteInfo *route_data_pass):
        PrimaryDialog (TRect (0,0,79,23), "Route Identification"),
        TWindowInit (initFrame) {

        // turn little box off
        flags ^=wfClose;
        route_data_ptr=route_data_pass;
        insert (new TLabel (TRect (2,2,25,3),
                "Identify your route!",0) );

        TButton *default_button=new TButton (TRect(1,4,65,6),
                ROUTE_NAME_ANAC,
                id_route_anac, bfNormal);

        insert (default_button);

        insert (new TButton (TRect (1,6,65,8), ROUTE_NAME_TOWN,
                id_route_town, bfNormal));
        insert (new TButton (TRect(1,8,65,10), ROUTE_NAME_MUKI,
                id_route_muki, bfNormal));

        insert (new TButton (TRect (1,10,65,12), ROUTE_NAME_EDMD,
                id_route_edmd, bfNormal));
        insert (new TButton (TRect(1,12,65,14), ROUTE_NAME_SEAW,
                id_route_seaw, bfNormal));

        insert (new TButton (TRect (1,14,65,16), ROUTE_NAME_SEAB,
                id_route_seab, bfNormal));

        insert (new TButton (TRect(1,16,65,18), ROUTE_NAME_FVSO,
                id_route_fvso, bfNormal));

        insert (new TButton (TRect (1,18,65,20), ROUTE_NAME_SEAV,
                id_route_seav, bfNormal));
        insert (new TButton (TRect(1,20,65,22), ROUTE_NAME_POIT,
                id_route_poit, bfNormal));
```

```
        default_button->select();


}
void RouteInit::handleEvent (TEvent & event) {
        PrimaryDialog::handleEvent (event);
        if (event.what==evCommand) {
                if(event.message.command >=id_route_start &&
                        event.message.command <=id_route_end) {
                        strncpy (route_data_ptr->route_name,
                                RouteNameLookup(event.message.command),
                                MaxRouteNameLength);


                        clearEvent(event);
                        event.what=evCommand;
                        event.message.command=cmCancel;
                        putEvent(event);
                }

        }
}

/****   crew initializer    ******/

CrewInit::CrewInit (CrewInfo *crew_data_pass):
        PrimaryDialog (TRect (0,0,79,23), "Crew Identification"),
        TWindowInit (initFrame) {

        // turn little box off
        flags ^=wfClose;

        crew_data_ptr=crew_data_pass;
        insert (new TLabel (TRect (2,2,25,3),
                "Identify your crew!",0) );
                // REMEMBER TO CHANGE position name consts in consts.h
                // if the below titles are changes

        // grab the default information from the file
        WatchCrewLoad(crew_data_ptr);



        TInputLine *Master =new TInputLine(TRect (2,5,30,6),
                crew_name_length);
        insert (new TLabel (TRect (2,4,30,5), "Enter Master's Name",Master) );
        insert (Master);

        TInputLine *CM= new TInputLine(TRect (2,8,30,9), crew_name_length);
        insert (new TLabel (TRect (2,7,30,8), "Enter Chief Mate's Name",CM) );
        insert (CM);

        TInputLine *SM =new TInputLine(TRect (2,11,30,12), crew_name_length);
        insert (new TLabel (TRect (2,10,30,11),
                "Enter Second Mate's Name",SM));
        insert (SM);

        TInputLine *QM =new TInputLine(TRect (2,14,30,15), crew_name_length);
        insert (new TLabel (TRect (2,13,30,14),
```

```
            "Enter QuarterMaster's Name",QM) );
        insert(QM);

        TInputLine *Bsn=new TInputLine(TRect (2,17,30,18), crew_name_length);
        insert (new TLabel (TRect (2,16,30,17), "Enter Bosun's Name",Bsn) );
        insert(Bsn);

        TInputLine *AB1=new TInputLine(TRect (2,20,30,21), crew_name_length);
        insert (new TLabel (TRect (2,19,30,20), "Enter AB #1's Name",AB1) );
        insert (AB1);

// second column
        TInputLine *AB2 =new TInputLine(TRect (33,5,63,6),crew_name_length);
        insert (new TLabel (TRect (33,4,63,5), "Enter AB #2's Name",AB2) );
        insert (AB2);

        TInputLine *OS1=new TInputLine(TRect (33,8,63,9), crew_name_length);
        insert (new TLabel (TRect (33,7,63,8), "Enter OS #1's Name",OS1) );
        insert (OS1);

        TInputLine *OS2=new TInputLine(TRect (33,11,63,12), crew_name_length);
        insert (new TLabel (TRect (33,10,63,11), "Enter OS #2's Name",OS2) );
        insert (OS2);

        TInputLine *OS3=new TInputLine(TRect (33,14,63,15), crew_name_length);
        insert (new TLabel (TRect (33,13,63,14), "Enter OS #3's Name",OS3) );
        insert(OS3);

        TInputLine *OS4=new TInputLine(TRect (33,17,63,18), crew_name_length);
        insert (new TLabel (TRect (33,16,63,17), "Enter OS #4's Name",OS4) );
        insert (OS4);

        TButton * Dbutton= new TButton (TRect (40,2,70,4),
            "Done!", cmYes, bfNormal);
        insert(Dbutton);

        setData(&crew_data_ptr->crew_name);
        Dbutton->select();

}


void CrewInit::handleEvent (TEvent & event) {

        if (event.what != evCommand ) PrimaryDialog::handleEvent (event);
        if (event.what==evCommand && event.message.command!=cmYes)
            PrimaryDialog::handleEvent(event);
        if (event.what==evCommand && event.message.command==cmYes) {
            //save crew data to structure
            getData((&crew_data_ptr->crew_name));

            //now save the data to the file
            WatchCrewSave(crew_data_ptr);
            clearEvent (event);
            // now close the box
            event.what=evCommand;
            event.message.command=cmCancel;
            putEvent(event);
        }
```

```
}


/*****   tides initializer   ****/
TidesInit::TidesInit (TidesInfo *tides_data_pass):
        PrimaryDialog (TRect (0,0,79,23), "Tides Identification"),
        TWindowInit (initFrame) {
        // turn little box off
        flags ^=wfClose;
        // experiment
        tides_data_ptr=tides_data_pass;
        insert (new TLabel (TRect (2,2,35,3),
                "Enter tide levels and times.",0) );
        insert (new TButton (TRect(37,2,65,4), "Ok, Done!",
                cmYes, bfNormal));


        insert (new TLabel (TRect (3,4,75,5),
        "High Tides   High Tides   High Tides   High Tides   High Tides    High Tides",0) );
        insert (new TLabel (TRect (2,6,28,7), "Enter Tide height",0) );
        insert (new TLabel (TRect (45,6,73,7), "Enter Tide Time",0) );

        TInputLine *default_line =new TInputLine(TRect (2,8,22,9),
                tides_length);
        insert (default_line);

        insert (new TLabel (TRect (30,8,44,9), "First",0) );
        insert (new TInputLine(TRect (45,8,65,9), tides_length));

        insert (new TInputLine(TRect (2,10,22,11), tides_length));
        insert (new TLabel (TRect (30,10,44,11), "Second",0) );
        insert (new TInputLine(TRect (45,10,65,11), tides_length));

        insert (new TInputLine(TRect (2,12,22,13), tides_length));
        insert (new TLabel (TRect (30,12,44,13), "Third",0) );
        insert (new TInputLine(TRect (45,12,65,13), tides_length));

        insert (new TLabel (TRect (3,14,75,15),
        "Low Tides   Low Tides   Low Tides   Low Tides   Low Tides    Low Tides",0) );

        insert (new TInputLine(TRect (2,16,22,17), tides_length));
        insert (new TLabel (TRect (30,16,44,17), "First",0) );
        insert (new TInputLine(TRect (45,16,65,17), tides_length));

        insert (new TInputLine(TRect (2,18,22,19), tides_length));
        insert (new TLabel (TRect (30,18,44,19), "Second",0) );
        insert (new TInputLine(TRect (45,18,65,19), tides_length));

        insert (new TInputLine(TRect (2,20,22,21), tides_length));
        insert (new TLabel (TRect (30,20,44,21), "Third",0) );
        insert (new TInputLine(TRect (45,20,65,21), tides_length));

        default_line->select();
}

void TidesInit::handleEvent (TEvent & event) {

        if (event.what != evCommand ) PrimaryDialog::handleEvent (event);
```

```
        if (event.what==evCommand && event.message.command!=cmYes)
                PrimaryDialog::handleEvent(event);
        if (event.what==evCommand && event.message.command==cmYes) {
                getData(tides_data_ptr);
                clearEvent (event);
                // now close the box
                event.what=evCommand;
                event.message.command=cmCancel;
                putEvent(event);
        }

}

/*****   initial arrival and departure class section  ********/


ArrivalInit::ArrivalInit(MoveInfo *arrival_chosen,
        RouteInfo *arrival_possible):PrimaryDialog( TRect(3,0,75,23),
        "Arrival Setup"), TWindowInit (initFrame) {

        flags ^=wfClose;  //eliminate the little box

        // copy data elements
        selected_data_ptr=arrival_chosen;
        possible_data_ptr=arrival_possible;
        int button;
        const int ButtonOffset=3;
        TButton *P_button[MaxRouteData_ad];

        insert (new TLabel ( TRect(3,1,62,2),
                "Enter the initial point of arrival for this end.",0));
        int end_loop = non_empty_fields();
        for (button=0; button <= end_loop; button++) {

                P_button[button] =new TButton (ButtonSizer(ButtonOffset,
                        MaxCheckLength, button+1 ),
                        possible_data_ptr->arrival[button],
                        cmArrivalChooseBt + button, bfNormal);
                insert(P_button[button]);

      P_button[0]->select();
        }

 }
 TRect ArrivalInit::ButtonSizer(int startbutton, int lwidth, int number){
        const int odd =0;
        const int even=1;
        const int bwidth=2;
        const int voffset=3;
        const int hoffset=3;

        int lx=0;int ly=0;
        int rx=0;int ry=0;

        int status=0;
        if (number % 2 ==0) status=even;
        else status=odd;
        if (status==odd) {
```

```
                lx=startbutton;
                ly= (number/2) *
                        (bwidth)+voffset;
                rx=lx +lwidth+4;
                ry=ly+bwidth;
        }
        else if (status==even) {
                lx =startbutton+lwidth+hoffset;
                ly = ((number-2)/2)*bwidth+voffset;
                rx=lx+lwidth+4;
                ry=ly+bwidth;
        }

        return (TRect(lx,ly,rx,ry) );
}

void ArrivalInit::handleEvent (TEvent &event) {

        PrimaryDialog::handleEvent(event);

        if (event.what==evCommand) {       /* this code determines which
                                arrival button was pushed */

                if (event.message.command >=cmArrivalChooseBt  &&
                        event.message.command <=cmArrivalChooseBt
                                +MaxRouteData_ad ) {
                                int touch_button=
                                  event.message.command-cmArrivalChooseBt;
                                strncpy(selected_data_ptr->arrival,
                                 possible_data_ptr->arrival[touch_button],
                                        MaxCheckLength);

                                // now signal end of dialog
                                clearEvent(event);
                                event.what=evCommand;
                                event.message.command=cmCancel;
                                putEvent(event);

                }
        }
}
int ArrivalInit::non_empty_fields(void) {
        int counter=0;
        for (counter =0; counter < MaxRouteData_ad; counter++) {
                if(possible_data_ptr->arrival[counter][0]=='\0')  break;
        }
        --counter;
        if (counter<=0) counter++;
        return counter;
}
// departure

DepartureInit::DepartureInit(MoveInfo *departure_chosen,
        RouteInfo *departure_possible):PrimaryDialog( TRect(3,0,75,23),
        "Departure Setup"), TWindowInit (initFrame) {

        flags ^=wfClose;  //eliminate the little box
```

```
        // copy data elements
        selected_data_ptr=departure_chosen;
        possible_data_ptr=departure_possible;
        int button;
        const int ButtonOffset=3;
        TButton *P_button[MaxRouteData_ad];

        insert (new TLabel ( TRect(3,1,62,2),
                "Enter the initial point of departure for this end.",0));
        int end_loop = non_empty_fields();
        for (button=0; button <= end_loop; button++) {

                P_button[button] =new TButton (ButtonSizer(ButtonOffset,
                        MaxCheckLength, button+1 ),
                        possible_data_ptr->depart[button],
                        cmDepartureChooseBt + button, bfNormal);
                insert(P_button[button]);

        P_button[0]->select();
        }

}
TRect DepartureInit::ButtonSizer(int startbutton, int lwidth, int number){
        const int odd =0;
        const int even=1;
        const int bwidth=2;
        const int voffset=3;
        const int hoffset=3;

        int lx=0;int ly=0;
        int rx=0;int ry=0;

        int status=0;
        if (number % 2 ==0) status=even;
        else status=odd;
        if (status==odd) {
                lx=startbutton;
                ly= (number/2) *
                        (bwidth)+voffset;
                rx=lx +lwidth+4;
                ry=ly+bwidth;
        }
        else if (status==even) {
                lx =startbutton+lwidth+hoffset;
                ly = ((number-2)/2)*bwidth+voffset;
                rx=lx+lwidth+4;
                ry=ly+bwidth;
        }

        return (TRect(lx,ly,rx,ry) );
}

void DepartureInit::handleEvent (TEvent &event) {

        PrimaryDialog::handleEvent(event);

        if (event.what==evCommand) {       /* this code determines which
                                        departure button was pushed */
```

```
                if (event.message.command >=cmDepartureChooseBt  &&
                        event.message.command <=cmDepartureChooseBt
                                +MaxRouteData_ad ) {
                                int touch_button=
                                  event.message.command-cmDepartureChooseBt;
                                strncpy(selected_data_ptr->depart,
                                 possible_data_ptr->depart[touch_button],
                                        MaxCheckLength);

                                // now signal end of dialog
                                clearEvent(event);
                                event.what=evCommand;
                                event.message.command=cmCancel;
                                putEvent(event);

                }
        }
}
int DepartureInit::non_empty_fields(void) {
        int counter=0;
        for (counter =0; counter < MaxRouteData_ad; counter++) {
                if(possible_data_ptr->depart[counter][0]=='\0')  break;
        }
        --counter;
        if (counter<=0) counter++;
        return counter;
}
```

```
// this file is guimov3.cpp and contains graphical classes for the
// movement and other views, including the TimeGrabber class

// class definition file for the tidesviever, crewviewer, and the movement
// subportions of the WSF Gui project


#define Uses_TDrawBuffer
#define Uses_TWindow
#define Uses_TEvent
#define Uses_TButton
//#define Uses_TView
//#define Uses_TApplication
//#define Uses_TStatusLine
//#define Uses_TMenuBar
//#define Uses_TDeskTop
//#define Uses_TStatusDef
//#define Uses_TStatusItem
//#define Uses_TSubMenu
//#define Uses_TMenuItem
#define Uses_TKeys
//#define Uses_TProgram
#define Uses_TInputLine
#define Uses_TLabel
#define Uses_TObject

// include files

#include "frrygui3.h"
#include "frrydat2.h"
#include <tv.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>
// #include <dos.h>  // for sound


// test global
//extern CrewInfo crewit;
//extern MoveInfo locations;
extern FerryDataCollection  FerryData_global;


// TidesViewer section
TidesViewer::TidesViewer(const TRect &r, short aNumber,
        TidesInfo tides_pass) :

        TWindow(r, "Tides Information", aNumber),
        TWindowInit(TidesViewer::initFrame) {

        TRect rect = getClipRect(); // get exposed area
        rect.grow(-1,-1);  //make interior fit the window
        insert (new TidesInterior (rect, tides_pass) );  // add interior to the window
```

```
            // copy passed values to the tides_data structure
        }
        void TidesViewer::handleEvent (TEvent &event) {
            TWindow::handleEvent(event); // act like a base ?
            switch (event.message.command) {
                case msgTidesViewer:
                    clearEvent (event);
                break;
            }
        }

        TidesInterior::TidesInterior( const TRect& bounds, TidesInfo &tides_pass) : TView(
bounds ) {
            // copy passed values to the tides_data structure

            int t_entry;
            for (t_entry=0 ; t_entry < tides_entries ; t_entry++) {

                strncpy(tides_data.high[t_entry].time,
                    tides_pass.high[t_entry].time,tides_length);

                strncpy (tides_data.high[t_entry].height,
                    tides_pass.high[t_entry].height,tides_length);

                strncpy(tides_data.low[t_entry].time,
                    tides_pass.low[t_entry].time,tides_length);

                strncpy (tides_data.low[t_entry].height,
                    tides_pass.low[t_entry].height,tides_length);

            }

            growMode = gfGrowHiX | gfGrowHiY;
              //make size follow that of the window
            options = options | ofFramed;
        }
        int TidesInterior::print(int *cur_line_ptr, char first[],
                    char second[]) {

            // size.x and size.y are inherited data structures that
            // give the size of the window being implemented.


            if (*cur_line_ptr >= size.y ) return Interior_Stop;
                                                    //cur_line_ptr points to
            else {                  // the current line drawn
                ushort color=getColor(0x0301);
                char append_string[tides_length*3];
                TDrawBuffer b;
                char s[maxLineLength];
                for (int y=0; y<tides_length*3; y++) {
                    append_string[y]='\0';
                }
                b.moveChar( 0, ' ', color, size.x );
                // fill line buffer with spaces
                strcpy(append_string, first);
                strcat(append_string, "    ");
                strcat(append_string, second);
                strncpy(s,append_string, size.x);
```

```
                s[size.x] = EOS;
                b.moveStr( 0, s, color );
                writeLine( 0, *cur_line_ptr, size.x, 1, b);
                (*cur_line_ptr)++;
                return Interior_OK;
        }
}
void TidesInterior::draw() {

        char tides_header[]="Tides   Time";
        char tides_high_header[]="High";
        char tides_low_header[]="Low";
        int line=0;
        int *line_ptr;
        line_ptr=&line;

        print (line_ptr, tides_header);
        print (line_ptr, tides_high_header );

        for (int entry=0; entry< tides_entries; entry++) {
                print (line_ptr, tides_data.high[entry].height,
                        tides_data.high[entry].time);
        }
        print (line_ptr,  tides_low_header );
        for (entry=0; entry < tides_entries; entry++) {
                print (line_ptr, tides_data.low[entry].height,
                        tides_data.low[entry].time);
        }

        while ( print (line_ptr,  " ")!=Interior_Stop)
                {//empty interior
        }

}
/************   Crew viewer section of the code *************/

/********************  CrewViewer class definition  ***************/
CrewViewer::CrewViewer( const TRect& bounds, short aNumber,
        CrewInfo &crew_pass) :
        TWindow( bounds, "Crew", aNumber),
        TWindowInit( CrewViewer::initFrame )    {

        TRect r = getClipRect();   // get exposed area
        r.grow(-1, -1);          // make interior fit inside window frame
        insert( new CrewViewerInterior(r, crew_pass) );
                // add interior to window
}

void CrewViewer::handleEvent (TEvent &event) {
        TWindow::handleEvent(event); //add base functions
        switch (event.message.command) {
                case msgCrewViewer:
                        clearEvent(event);
                break;
        }
}

/********************  CrewViewerInterior class definition ****************************/
```

```
CrewViewerInterior::CrewViewerInterior( const TRect& bounds,
        CrewInfo &crew_pass) : TView( bounds ) {

        // copy the passed values into a member structure
        int looper=0;
        for (looper=0; looper < crew_positions; looper++) {
                strncpy (crew_data.crew_name[looper],
                        crew_pass.crew_name[looper],
                        crew_name_length);
                strncpy (crew_data.crew_position_name[looper],
                        crew_pass.crew_position_name[looper],
                        crew_name_length);
        }
        growMode = gfGrowHiX | gfGrowHiY;
          //make size follow that of the window
        options = options | ofFramed;
}

void CrewViewerInterior::draw() {
        int line=0;
        int *line_ptr;
        line_ptr=&line;
        int looper;
        char long_line[crew_name_length*2+10];
        strncpy (long_line, " " , crew_name_length*2+10);

        //feed the data to the handy class print routine

        for (looper=0; looper < crew_positions; looper+=2) {
                int inc_looper;
                inc_looper=looper+1;
                if (inc_looper >=crew_positions)
                        inc_looper=crew_positions-1;

                strsetup (long_line,crew_data.crew_position_name[looper],
                        crew_data.crew_position_name[inc_looper]);
                print (line_ptr,long_line);

                strsetup (long_line, crew_data.crew_name[looper],
                        crew_data.crew_name[inc_looper]);
                print (line_ptr, long_line);
        }
        while (print (line_ptr, " ")!=Interior_Stop) {
        // empty function interior
        }
}
char * CrewViewerInterior::strsetup( char *outstr, char * instr1,
        char *instr2) {
        const int MaxStrLength=34; // must be less than length of string
        const int SpaceDis=5;
        int i;
        for (i=0; i < MaxStrLength; i++) {
                if (instr1[i]=='\0') break;
                outstr[i]=instr1[i];
        }
        for (i; i < MaxStrLength; i++) {
                outstr[i]=' ';
        }
```

```
        for (i; i < MaxStrLength+SpaceDis; i++) {
                outstr[i]=' ';
        }
        for (i; i < MaxStrLength*2+SpaceDis; i++) {
                outstr[i]=instr2[i-(MaxStrLength+SpaceDis)];
        }
        return outstr;
}


int CrewViewerInterior::print (int *cur_line_ptr, char first[]) {
        // the cur_line_ptr points to the current line being drawn


        if (*cur_line_ptr >= size.y) return Interior_Stop;

        else {                          // the current line drawn
                ushort color=getColor(0x0301);
                char append_string[crew_name_length*2+10];
                TDrawBuffer b;
                char s[maxLineLength];
                for (int y=0; y<crew_name_length*2+10; y++) {
                        append_string[y]='\0';
                }
                b.moveChar( 0, ' ', color, size.x );
                // fill line buffer with spaces
                strcpy(append_string, first);
          //    strcat(append_string, "    ");    //not needed for this
          //    strcat(append_string, second);    // classes data
                strncpy(s,append_string, size.x);

                s[size.x] = EOS;
                b.moveStr( 0, s, color );
                writeLine( 0, *cur_line_ptr, size.x, 1, b);
                (*cur_line_ptr)++;
                return Interior_OK;
        }
}


/*****   movement class section  ********/

MovementViewer::MovementViewer(MoveInfo chk_data):
        PrimaryDialog( TRect(20,0,65,19), "Movement"),
        TWindowInit (initFrame) {
        // copy data elements

        int i;

        const int ButtonOffset=6;
        chkpoint_data.number_of_points=chk_data.number_of_points;
        strncpy (chkpoint_data.arrival, chk_data.arrival,
                MaxCheckLength);
        strncpy (chkpoint_data.depart, chk_data.depart, MaxCheckLength);
        for (i=0; i < MaxCheckPoints-2; i++) {
                strncpy(chkpoint_data.chkpoint[i], chk_data.chkpoint[i],
                        MaxCheckLength);
        }
```

```
        TButton *DE_button=new TButton (TRect(20,2,35,4),
                "Departure", cmDeparture, bfNormal);
        insert (new TLabel(TRect(5,2,17,3),chkpoint_data.depart,
                DE_button));

        insert (DE_button);
        for (i=0; i< chkpoint_data.number_of_points; i++) {
                insert (new TButton (ButtonSizer (ButtonOffset,
                        MaxCheckLength, i+3), chkpoint_data.chkpoint[i],
                        ButtonPopBottom+i, bfNormal));

        }

        TButton *AR_button=new TButton (TRect(20,15,35,17),
                "Arrival", cmArrival, bfNormal);
        insert (new TLabel(TRect(5,15,17,16),chkpoint_data.arrival,
                AR_button));
        insert (AR_button);
        DE_button->select();

}
TRect MovementViewer::ButtonSizer(int startbutton, int lwidth, int number){
        // this routine places movement buttons dynamically.
        // this is to allow using different numbers of buttons
        // for different routes

        const int odd =0;
        const int even=1;
        const int bwidth=2;
        const int voffset=3;
        const int hoffset=3;

        int lx=0;int ly=0;
        int rx=0;int ry=0;

        int status=0;
        if (number % 2 ==0) status=even;
        else status=odd;
        if (status==odd) {
                lx=startbutton;
                ly= (number/2) *
                        (bwidth)+voffset;
                rx=lx +lwidth+4;
                ry=ly+bwidth;
        }
        else if (status==even) {
                lx =startbutton+lwidth+hoffset;
                ly = ((number-2)/2)*bwidth+voffset;
                rx=lx+lwidth+4;
                ry=ly+bwidth;
        }

        return (TRect(lx,ly,rx,ry) );
}

void MovementViewer::handleEvent (TEvent &event) {

        PrimaryDialog::handleEvent(event);
        if (event.what==evBroadcast) {
```

```
                switch (event.message.command) {
                        case msgMovementViewer:
                                clearEvent(event);
                        break;
                }
        }
        if (event.what==evCommand) {       /* this code determines which
                                checkpoint button was pushed */
                /*
                The selectNext(False) command allows the buttons to
                Automatically cycle forward thru the list */
                /* the false is counterintuitive, but is the
                TVision library requirement */

                if (event.message.command >=ButtonPopBottom &&
                        event.message.command <=ButtonPopTop) {
                        int button_number=0;
                        button_number = event.message.command -
                                ButtonPopBottom;
                                clearEvent(event);
                        myChkpointConfirm
                                (button_number);
                                selectNext(False);
                                // calls up a chkpoint time confirm window
                }
                else if (event.message.command==cmArrival){
                        clearEvent(event);
                        myArrivalConfirm();
                        selectNext(False);
                }
                else if (event.message.command==cmDeparture){
                        clearEvent(event);
                        myDepartConfirm();
                        selectNext(False);
                }
        }
}
void MovementViewer::myArrivalConfirm(void) {

        MvTimeConfirm *timefirm=new
                        MvTimeConfirm(chkpoint_data.arrival,
                                ACTION_ARRIVE);
                this->execView (validView(timefirm));
                destroy (timefirm);
}
void MovementViewer::myDepartConfirm(void) {

        MvTimeConfirm *timefirm=new
                        MvTimeConfirm(chkpoint_data.depart,
                                ACTION_DEPART);
                this->execView (validView(timefirm));
        destroy (timefirm);
}
void MovementViewer::myChkpointConfirm(int chkpoint_num) {

                MvTimeConfirm *timefirm=new
                        MvTimeConfirm(chkpoint_data.chkpoint[chkpoint_num],
                        ACTION_CHKPOINT);
                this->execView (validView(timefirm));
```

```
        destroy (timefirm);
}


/*****  classes that movement viewer uses ************/

/* movement time confirm modal dialog  */

MvTimeConfirm::MvTimeConfirm(char Cpoint[MaxCheckLength],
        const char pass_type[MaxCommandLength]):
        SecondaryDialog (TRect (2,2,35,17),"How Long Ago?"),
        TWindowInit (initFrame) {

        // initialize occur time
        time_info.machineTime=0L;
        time_info.givenTime=0L;
        // initialize occur point
        strncpy(GeoPoint, Cpoint, MaxCheckLength);

        strncpy (point_type, pass_type,MaxCommandLength);

        insert (new TLabel (TRect(4,2,30,3), "When did we reach /leave:",0));
        insert (new TLabel (TRect(2,4,17,5), GeoPoint, 0));

        TButton * JN= new TButton (TRect(2,7,15,9), "Just Now", cmOccurNow,
                bfDefault);
        insert (JN);
        insert (new TButton (TRect(17,7,28,9), "Earlier", cmOccurBefore,
                bfNormal));
        insert (new TButton (TRect(2,10,25,12),
                "OOps! Go Back", cmCancel, bfNormal));

        JN->select();

}
void MvTimeConfirm::handleEvent(TEvent &event) {
        // time_t time_out;
        if (event.what!=evCommand) SecondaryDialog::handleEvent(event);
        if (event.what==evCommand && event.message.command==cmCancel)
                SecondaryDialog::handleEvent(event);

        if (event.what==evCommand && event.message.command!=cmCancel) {
                switch(event.message.command) {
                        case cmOccurNow:
                                clearEvent(event);

                                // get the time of the button press
                                // time_out=time (NULL);
                                time_info.givenTime=
                                        time_info.machineTime=
                                        time(NULL);

                                // facility for closing up the window
                                event.what = evCommand;
                                event.message.command = cmClose;
                                putEvent( event );


                        break;
```

```
                        case cmOccurBefore:
                                clearEvent(event);
                                // get the time of the button press
                          //    time_out=time (NULL);
                                time_info.givenTime=
                                        time_info.machineTime=
                                        time(NULL);

                                int status=0;
                                // checks the status of the
                                // time grabber window
                                // valid if altered,
                                // invalid if oops is pressed


                                TimeGrabber *timeWindow =
                                        // below arg changed from pointer
                                        new TimeGrabber (&time_info, &status);
                                this ->execView(validView(timeWindow));
                        destroy (timeWindow);
                                // Now close this window if data was valid
                                if (status==VALID) {
                                        data_output();
                                        // now close the window
                                        event.what = evCommand;
                                        event.message.command = cmCancel;
                                        putEvent( event );
                                }

                        break;
                        case cmClose:
                                clearEvent(event);
                                data_output();
                                // now close the window
                                event.what = evCommand;
                                event.message.command = cmCancel;
                                putEvent( event );
                        break;

                }
        }
}
void MvTimeConfirm::data_output(void) {

        MakeASound(sound_record);
        MovementDataPacket *MDP= new MovementDataPacket(GeoPoint,point_type,
                time_info);
        assert (MDP!=NULL);
        FerryData_global.insert(MDP);


}

TimeGrabber::TimeGrabber( OccurTime *time_info, int *status):
        SecondaryDialog (TRect (0,0,30,15),"When?"),
        TWindowInit (initFrame) {

        strncpy(inputtime, " ",6); //initalize inputtime
        time_ptr=time_info;
```

```
        status_ptr=status;  // point the member pointer to the passed
                            // value



        insert (new TButton (TRect(2,10,25,12), "OK", cmOK, bfNormal));

        insert (new TButton (TRect(2,12,25,14),
                "OOps!  Go Back", cmCancel, bfNormal));

        selectNext(False);
        TGTime.TimeDisplaySet(time_ptr->givenTime);
        strncpy(inputtime, TGTime.FormatVert(),6);


        TInputLine *timeLine=new TInputLine(TRect(6,4,14,5),6);
        insert (timeLine);
        insert (new TButton (TRect(2,6,14,8),"UP", cmUPTIME, bfNormal));
        insert (new TButton (TRect(16,6,25,8), "Down", cmDOWNTIME,
                bfNormal));

        setData(inputtime);

}
void TimeGrabber::draw(void){
        strncpy(inputtime, TGTime.FormatVert(),6);
        setData(inputtime);
        TDialog::draw();
}
void TimeGrabber::handleEvent (TEvent &event) {

        if (event.what!=evCommand) SecondaryDialog::handleEvent(event);
        if (event.what==evCommand && (event.message.command!=cmNo &&
                event.message.command!=cmOK))
                SecondaryDialog::handleEvent(event);
        if (event.what==evCommand &&event.message.command==cmUPTIME) {
                clearEvent(event);
                TGTime.Inc();
                drawView();
        }
        if (event.what==evCommand && event.message.command==cmDOWNTIME) {
                clearEvent(event);
                TGTime.Dec();
                drawView();
        }
        if (event.what==evCommand && event.message.command==cmOK) {

                //getData(inputtime);
                clearEvent(event);
                if (checkData()==VALID) {   // valid data returns a 1
                        // close itself
                        *status_ptr=VALID;
                        event.what = evCommand;
                        event.message.command = cmCancel;
                        putEvent( event );
                }

        }
        if (event.what==evCommand && event.message.command==cmNo) {
```

```
                clearEvent(event);
                *status_ptr=INVALID;
                event.what = evCommand;
                event.message.command = cmCancel;
                putEvent( event );

        }

}
int TimeGrabber::checkData(void) {
        // function to check the data for validity
        int invalid_flag=0;
        int weird_flag=0;
        int Modify_flag=0;
        char i_minutes[3];  // conversion on input time to minutes and seconds
        char i_hours[3];
        char m_minutes[3];//conversion of machine time to minutes and seconds
        char m_hours[3];
        char longtime[26];
        char machine_time[6];
        long gotten_time_seconds;
        long machine_time_seconds;
        long time_diff=0L;

        // check each element for correct data type
        if(inputtime[0] < 48 || inputtime[0] >57) invalid_flag=1;
        //check for a number
        if(inputtime[1] < 48 || inputtime[1] >57) invalid_flag=1;
        if (inputtime[2]!=':') invalid_flag=1;  //check for colon
        if(inputtime[3] < 48 || inputtime[3] >57) invalid_flag=1;
        if(inputtime[4] < 48 || inputtime[4] >57) invalid_flag=1;

        // now check for improper number entries
        if (invalid_flag!=1) {
                sscanf(inputtime, "%2s%*1c%2s", i_hours, i_minutes);
                if (atoi(i_hours) <0 || atoi(i_hours) >23) invalid_flag=1;
                if(atoi(i_minutes) <0 || atoi(i_minutes) > 59) invalid_flag=1;
        }
        // now check for weird entries
        // check to see if given time is later than machine time
        //this could happen if it is just past midnight
        //and an event happened prior to midnight
        if (invalid_flag!=1) {
                //convert given time to seconds since midnight

                gotten_time_seconds= 3600L* (atol(i_hours))
                        +(60L *(atol(i_minutes)) );

                // now convert machine time to seconds since midnight
                strcpy (longtime, ctime (&(time_ptr->machineTime)) );
                sscanf (&longtime[11], "%5s", machine_time);
                sscanf(machine_time, "%2s%*1c%2s",
                        m_hours, m_minutes);
                machine_time_seconds= 3600L*(atol(m_hours))+
                        (60L *(atol(m_minutes)) );

                if (machine_time_seconds < gotten_time_seconds)
```

```
                    weird_flag=1;
        }


        if(invalid_flag==1) {
                // call a report box to report the invalid entry
                BadTimeDialog *bad_time = new BadTimeDialog();
                this ->execView (validView(bad_time));
                destroy (bad_time);
        }
        if (invalid_flag==0 && weird_flag==0) {
                time_diff= machine_time_seconds-gotten_time_seconds;
                time_ptr->givenTime=(time_ptr->machineTime) -time_diff;
        }
        if (invalid_flag==0 && weird_flag==1) {
                // create a box to confirm choice
                WeirdTimeDialog *weird_time =new WeirdTimeDialog
                        (&weird_flag);
                this ->execView (validView(weird_time));
                destroy (weird_time);
                Modify_flag=1;  // note that data was weird and needs
                                // seperate processing
        }
        if (weird_flag==0 && Modify_flag==1) {
                // now process weird time
                if (atoi(i_hours)==23 && atoi(m_hours)==0) {
                        // we have flipped a day, so we need to handle
                        // in an appropriate manner
                 // find time as if it was normal and then subtract
                 // a full day
                time_diff= machine_time_seconds-gotten_time_seconds;
                time_ptr->givenTime=(time_ptr->machineTime) -time_diff
                - 86400L;  // seconds in a day

                }
                else {  // the time is simply later for one reason or
                        // another
                        time_diff=gotten_time_seconds-machine_time_seconds;
                        time_ptr->givenTime=
                                (time_ptr->machineTime)+time_diff;
                }
        }

        if (invalid_flag!=1 && weird_flag!=1) return VALID;
        else return INVALID;
}

BadTimeDialog::BadTimeDialog(void):SecondaryDialog(TRect (1,1,27,15),
                        "Invalid Entry"),TWindowInit(initFrame){


        insert (new TLabel (TRect(2,2,18,3),
                "Time given is ",0));
        insert (new TLabel (TRect (3,3,18,4),
                "incorrect.",0));
        insert (new TLabel (TRect(2,5,18,6),
                "Please input a" ,0));
        insert (new TLabel (TRect (3,6,18,7),
                "valid number.",0));
```

```
        insert(new TButton (TRect(2,8,14,10),
                "OK", cmOK, bfDefault));

        // make a mistake alarm
        MakeASound(sound_oops);

}
WeirdTimeDialog::WeirdTimeDialog(int *w_flag):SecondaryDialog(TRect (1,1,27,15),
                        "Strange Entry"),TWindowInit(initFrame){

        w_ptr=w_flag;

        insert (new TLabel (TRect(2,2,18,3),
                "Time given is ",0));
        insert (new TLabel (TRect (3,3,18,4),
                "later than ",0));
        insert (new TLabel (TRect(2,4,18,5),
                "current time" ,0));

        insert (new TLabel (TRect (3,6,18,7),
                "Are you sure?",0));

        // may want to put in  more info
        insert(new TButton (TRect(2,7,14,9),
                "Yes", cmYes, bfDefault));
        insert (new TButton (TRect (2,9,14,11),
                "No", cmNo, bfDefault));


        // make a mistake alarm
        MakeASound(sound_oops);

}
void WeirdTimeDialog::handleEvent (TEvent &event) {

        if (event.what!=evCommand) SecondaryDialog::handleEvent(event);
        if (event.what==evCommand && (event.message.command!=cmYes
                && event.message.command!=cmNo))
                SecondaryDialog::handleEvent(event);


        if (event.what==evCommand && event.message.command==cmYes) {

                *w_ptr=0;  // set the weird flag to off

                // close itself
                event.what = evCommand;
                event.message.command = cmCancel;
                putEvent( event );
        }
        if (event.what==evCommand && event.message.command==cmNo) {

                // close itself
                event.what = evCommand;
                event.message.command = cmCancel;
                putEvent( event );
        }

}
```

GUIWX.CPP

```
// This file is guiwx.cpp and is used in the ferrylog program
// This file defines the weather data entry class.  The weather class
// itself (WeatherInfo) is defined in frrydat2.h and frrydat2.cpp


#include<assert.h>
#include "guiwx.h"
#define Uses_TLabel
#define Uses_TButton
#define Uses_TEvent
#include <tv.h>

extern FerryDataCollection  FerryData_global;


WeatherReport::WeatherReport (void) : PrimaryDialog(TRect (0,0,69,22),
        "Weather Information"), TWindowInit(initFrame) {

        flags^=wfClose; // box off
        weatherTime.machineTime=weatherTime.givenTime=0L;

        insert (new TLabel (TRect(2,2,45,3),
                "Enter all weather information of interest",NULL) );


        TInputLine *Clouds=new TInputLine(TRect(22,4,32,5),MaxWeatherLength);
        insert(new TLabel (TRect(2,4,20,5), "Clouds",Clouds));
        insert (Clouds);

        TInputLine *Precip=new TInputLine(TRect(22,6,32,7),
                MaxWeatherLength);
        insert (new TLabel (TRect(2,6,20,7), "Precipitation",Precip));
        insert (Precip);

        TInputLine *Vis=new TInputLine(TRect(22,8,32,9),MaxWeatherLength);
        insert(new TLabel (TRect(2,8,20,9), "Visibility",Vis));
        insert(Vis);

        TInputLine *WindD=new TInputLine(TRect(22,10,32,11),
                MaxWeatherLength);
        insert (new TLabel (TRect(2,10,20,11), "Wind Direction",WindD));
        insert(WindD);

        TInputLine *WindS=new TInputLine(TRect(22,12,32,13),MaxWeatherLength);
        insert(new TLabel (TRect(2,12,20,13), "Wind Speed",WindS));
        insert(WindS);

        TInputLine *Sea=new TInputLine(TRect(22,14,32,15),
                MaxWeatherLength);
        insert (new TLabel (TRect(2,14,20,15), "Sea Condition",Sea));
        insert(Sea);

        TInputLine *Temperature=new TInputLine(TRect(22,16,32,17),
                MaxWeatherLength);
        insert(new TLabel (TRect(2,16,20,17), "Temperature",Temperature));
```

```
        insert(Temperature);

        TInputLine *Barom=new TInputLine(TRect(22,18,32,19),
                MaxWeatherLength);
        insert (new TLabel (TRect(2,18,20,19), "Barometer",Barom));
        insert(Barom);

        insert(new TButton(TRect( 38,4,60,6),"Record Weather",
                cmWeatherRecord, bfNormal));
        insert(new TButton(TRect (38,18,60,20),"Junk this screen",
                cmWeatherJunk, bfNormal));

        Clouds->select();
        setData(&weatherData);

}



void WeatherReport::handleEvent (TEvent &event) {
        PrimaryDialog::handleEvent(event);


        if (event.what!=evCommand) PrimaryDialog::handleEvent(event);
        if (event.what==evCommand && event.message.command==cmCancel)
                PrimaryDialog::handleEvent(event);

        if (event.what==evCommand && event.message.command!=cmCancel) {
                switch(event.message.command) {
                        case cmWeatherRecord:  // accept information,
                                                // close window
                                clearEvent(event);
                                // get weather data
                                getData(&weatherData);
                                // get the time of the button press

                                weatherTime.givenTime=
                                        weatherTime.machineTime=
                                        time(NULL);
                                        // put data in packet
                                        WeatherDataPacket *WDP=
                                          new WeatherDataPacket(weatherData,
                                            weatherTime);
                                        assert (WDP!=NULL);
                                        FerryData_global.insert(WDP);


                                        // now close the window
                                        close();

                        break;

                        case cmWeatherJunk: // reject information,
                                                // close window
                                clearEvent(event);
                                // now close the window
                                close();
```

```
                    break;
                }

            }
        }
```

GUIWX.H

```
// this file is guiwx.h

/* this file contains a class declaration for the weather data entry
        view */


#include "myvis2.h"
#include "frrydat2.h"

class WeatherReport: public PrimaryDialog {
public:
        WeatherReport(void);
        virtual void handleEvent (TEvent &);
        WeatherInfo weatherData;
        OccurTime weatherTime;
};
```

MAKEFLOG.BAT

```
make -fferrylog.mak
del *.obj
make -fcombine.mak
del *.obj

rem object files (*.obj) must be removed, since ferrylog and combine
rem both rely on the same source files, with definitions
```

MISCDAT.H

```
// header file to define miscellaneous data structures as needed
// this file is miscdat.h

const int MaxReturnLength=70;
struct {
        int FlopStatusInt;
        char FlopStatusTxt[MaxReturnLength];
} typedef FlopInfo;
```

```
// This file is misgui1.cpp and is used by the ferrylog program
/* This file contains GUI classes for the ferrylog program, mostly those
relating to print options and warnings */

/* class bodies for misc. Ferry log Gui elements */
#include "misgui1.h"
#include "consts.h"
#include "c_func2.h"
#define Uses_TRadioButtons
#define Uses_TSItem
#define Uses_TRect
#define Uses_TLabel
#define Uses_TButton
#define Uses_TKeys
#define Uses_MsgBox
#define Uses_TEvent
#include <tv.h>
#include <string.h>


// tvision class for printer options



PrintOptions::PrintOptions (void):
        PrimaryDialog (TRect (10,3,49,23), "Print options"),
        TWindowInit (initFrame) {

        // turn little box off
        flags ^=wfClose;
        this->showMarkers=False;

        insert (new TLabel (TRect (2,2,25,3),
                "Print as we go...",0) );

        TRadioButtons *spew =new TRadioButtons (TRect (4,5,15,7),
                new TSItem("~O~ff",
                new TSItem("o~N~",0) ) );
        insert (spew);

        insert (new TLabel (TRect (2,9,25,10),
                "Print final report....",0) );

        TRadioButtons *final =new TRadioButtons (TRect (4,12,15,14),
                new TSItem("Off",
                new TSItem("On",0) ) );
        insert (final);


        insert (new TButton (TRect(2,16,12,19), "Ok" , cmOK, 0));
        insert (new TButton(TRect(16,16,26,19), "Cancel", cmCancel, 0));

        spew->select();
```

```
}
void PrintOptions::handleEvent (TEvent & event) {
        PrimaryDialog::handleEvent(event);

}
PrinterError::PrinterError (const char *errorMsg):
        PrimaryDialog (TRect (5,5,50,20), "WARNING!!!!"),
        TWindowInit (initFrame) {
        showMarkers=False;
        flags^=wfClose;

        // make an alert sound

        insert (new TLabel (TRect(4,4,34,5), "Printer Problem: ",0));
        insert (new TLabel (TRect(4,6,34,7),errorMsg,0));

        insert (new TLabel (TRect(2,8,44,9),
                "Either Fix printer or cancel printing",0));

        TButton *fix =new TButton (TRect(6,12,22,14),"Fix Printer", cmOK,
                bfNormal);
        insert (fix);

        TButton *quit =new TButton (TRect(25,12,35,14), "Cancel", cmCancel,
                bfNormal);
        insert (quit);
        fix->select();
        MakeASound(sound_alert);
}
```

MISGUI1.H

```
/* this file contains miscellaneous gui classes for the ferry log program */
// this file is misgui1.h
/* this file contains class declaration information */

#include "frrygui3.h"
#define Uses_TWindow
#define Uses_TScroller
#include <tv.h>

class PrintOptions: public PrimaryDialog {
public:
        PrintOptions(void);
        virtual void handleEvent (TEvent &);
};

        //DATA from printOptions view
class PrintOptionData {
public:
        ushort spewData;
        ushort finalData;
        PrintOptionData (void) {spewData=0; finalData=0;}

};
class PrinterError:public PrimaryDialog {
public:
        PrinterError (const char *errorMsg);
//      void handleEvent(TEvent &);
};
/*
class LogDataInterior : public TScroller
{

public:

        LogDataInterior( const TRect& bounds, TScrollBar *aHScrollBar,
                TScrollBar *aVScrollBar );      // constructor
        virtual void draw();        // override TView::draw
private:
        void readFile(void);
};

class LogDataWindow : public TWindow                    // define a new window class
{

public:

        LogDataWindow(const TRect& bounds,const char *aTitle,short aNumber );
        LogDataInterior *makeInterior( const TRect& r);
  //    virtual void sizeLimits( TPoint& minP, TPoint& maxP );
        // override TWindow::sizeLimits

private:

        LogDataInterior *TheInterior;

};
*/
```

MYCOPY.CPP

```
// this is mycopy.cpp and is used in the ferrylog programs
// this subroutine is used to copy a file using a character by character
/* copy algorithm. This routine exists in an attempt to avoid
the problems associated with using system() on the compaq's , which
tends to hang the machine. This remedy has not completely worked,
since the Compaq still hangs on exit. If this problem can be remedied,
the driver3.cpp file could be modified to use the system command, and
this function could be eliminated */



#include <stdio.h>
#include <string.h>
#include <stdlib.h>


int mycopy (char * infilename, char *outfilename) {
        int c;
        FILE *inputfile;
        FILE *outputfile;


        if ( (inputfile=fopen (infilename, "r" )) !=NULL) {
                if ( (outputfile=fopen (outfilename, "w")) !=NULL) {
                        while ( ( c=fgetc (inputfile)) !=EOF) {
                                fputc( c, outputfile);

                        }
                        fclose (inputfile);
                        fclose (outputfile);

                 }
                else {
                        fprintf (stderr,"\n Can't open the output file %s\n",
                                outfilename);
                        exit (-1);
                }
        }
        else {
                fprintf (stderr, "\n Can't open the input file %s \n" ,
                        infilename);
                exit (-1);
        }
        return 0;
}
```

```
// this file is myvis2.cpp and is used in the ferrylog program
/* this file contains objects derived (inherited)
from standard TVision objects, and
was done to establish a standard for this program so that subsequent
derivations would be simpler */

#define Uses_TDialog
#define Uses_TInputLine
#define Uses_MsgBox
#define Uses_TKeys
#define Uses_TEvent

#include "myvis2.h"
#include <tv.h>

SecondaryDialog::SecondaryDialog(TRect &bounds, const char *aTitle)
        :TDialog (bounds, aTitle), TWindowInit(initFrame) {
        flags^=wfClose;
        showMarkers=True;
}

// copied from the handleEvent section of the source code for TDialog
// done in order to remove the special meaning that ESc key has for closing
// dialog boxes
void SecondaryDialog::handleEvent (TEvent &event) {

        TWindow::handleEvent(event);
        switch (event.what)     {
                case evKeyDown:
                        switch (event.keyDown.keyCode) {
                                case kbEsc:
                                        // event.what = evCommand;
                                        // event.message.command = cmCancel;
                                        // event.message.infoPtr = 0;
                                        // putEvent(event);
                                        // clearEvent(event);
                                break;
                                case kbEnter:
                                        event.what = evBroadcast;
                                        event.message.command = cmDefault;
                                        event.message.infoPtr = 0;
                                        putEvent(event);
                                        clearEvent(event);
                                break;
                        }
                        break;

                        case evCommand:
                                switch( event.message.command ) {
                                        case cmOK:
                                        case cmCancel:
                                        case cmYes:
                                        case cmNo:
                                        if( (state & sfModal) != 0 )            {
                                                endModal(event.message.command);
                                                clearEvent(event);
```

```
                                        }
                                        break;
                                }
                        break;
                }
        }

        TPalette& SecondaryDialog::getPalette(void) const {
                const char cpDoubleDialog[]=
        "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F\x10\x11\x12\x13\
x14\x15\x16\x17\x18\x19\x1A\x1B\x1C\x1D\x1E\x1F";
                static TPalette palette (cpDoubleDialog, sizeof(cpDoubleDialog)-1);
                return palette;
        }
        TView* SecondaryDialog::validView(TView* p) {
        // copied from distribution TProgram::validView
                if( p == 0 ) return 0;

                if( lowMemory() ) {
                        destroy( p );
                        outOfMemory();
                        return 0;
                }

                if( !p->valid( cmValid ) ) {
                        destroy( p );
                        return 0;
                }
                return p;
        }

        void SecondaryDialog::outOfMemory (void) {
                messageBox("Not enough memory to complete operation.", mfError |
                        mfOKButton);
        }
        // primary dialog
        PrimaryDialog::PrimaryDialog(TRect &bounds, const char *aTitle)
                :TDialog (bounds, aTitle), TWindowInit(initFrame) {
                showMarkers=True;
        }
        void PrimaryDialog::handleEvent (TEvent &event) {

                TWindow::handleEvent(event);
                switch (event.what)     {
                        case evKeyDown:
                                switch (event.keyDown.keyCode) {
                                        case kbEsc:
                                                // event.what = evCommand;
                                                // event.message.command = cmCancel;
                                                // event.message.infoPtr = 0;
                                                // putEvent(event);
                                                // clearEvent(event);
                                        break;
                                        case kbEnter:
                                                event.what = evBroadcast;
                                                event.message.command = cmDefault;
                                                event.message.infoPtr = 0;
                                                putEvent(event);
                                                clearEvent(event);
```

```
                        break;
                    }
                    break;

                    case evCommand:
                        switch( event.message.command ) {
                            case cmOK:
                            case cmCancel:
                            case cmYes:
                            case cmNo:
                            if( (state & sfModal) != 0 )        {
                                endModal(event.message.command);
                                clearEvent(event);
                            }
                            break;
                        }
                    break;
                }
}

TView* PrimaryDialog::validView(TView* p) {
// copied from distribution TProgram::validView

        if( p == 0 ) return 0;

        if( lowMemory() ) {
                destroy( p );
                outOfMemory();
                return 0;
        }

        if( !p->valid( cmValid ) ) {
                destroy( p );
                return 0;
        }
        return p;
}

void PrimaryDialog::outOfMemory (void) {
        messageBox("Not enough memory to complete operation.", mfError |
                mfOKButton);
}

myInputLine::myInputLine(TRect &bounds, int aMaxLen)
        :TInputLine(bounds,  aMaxLen) {

        // release line selection
        //selectAll(False);
        // set to insert mode in TInputLine



        //setState(sfCursorIns, Boolean(!(state & sfCursorIns)));

  //    TInputLine::TInputLine(bounds,  aMaxLen);
}
```

MYVIS2.H

```
// this file is myvis2.h
/* this file contains class declarations for basic inheritances from standard
Turbo vision classes */


#if!defined(__MYVIS1_H)
#define __MYVIS1_H


#define Uses_TDialog
#define Uses_TInputLine
#include <tv.h>

// secondary dialog is a dialog that is popped on top of another dialog
class SecondaryDialog: public TDialog {
public:
        SecondaryDialog(TRect &bounds, const char *aTitle);
        TView* validView(TView* p);
        void outOfMemory (void);
        virtual void handleEvent(TEvent &event);
private:
        TPalette& getPalette(void) const;
};
// primary dialogs are those dialogs that are inserted on the desktop
// directly, not on top of another dialog.
// the main difference is that the secondary dialog box has a different
// get palette than the primary dialog.

class PrimaryDialog: public TDialog {
public:
        PrimaryDialog(TRect &bounds, const char *aTitle);
        TView* validView(TView* p);
        void outOfMemory (void);
        virtual void handleEvent(TEvent &event);
};
class myInputLine: public TInputLine{

public:   // allows for insert mode as default
        myInputLine(TRect &bounds, int aMaxLen);
};

#endif // __MYVIS1_h
```

PSTREAM.CPP

```
// This file is pstream.cpp, and is used to build the ferrylog program

// this file was written by Pat Reilly
// on 10/10/92
//  this is a public domain set of classes that was obtained from
// Compuserve. For reference, the author's compuserve id is 70274,161

/* The compressed archive that this file was obtained from, PRNSTR.ZIP
was obtained from the Borland forum on Compuserve.  PRNSTR.ZOO, a recompressed
archive of the PRNSTR.ZIP file, is
included on the software diskette.  For more information about this library,
please see the header file and the read.me file contained in PRNSTR.ZOO
*/

/* This library was used to drive the printer  attached to LPT1: because
opening LPT1: by using a ofstream class didn't work well.  Using this class
made the printer work much better.  */


#include "PStream.h"
#include <Bios.h>

prnbuf::prnbuf()
        {
        xfd = EOF;
        isBinary = 0;
        char* p = new char[B_size];
        if( p )
                {
                setb( p, p+B_size,1 );    // ~streambuf() will delete.
                setp(base(),ebuf());
                setg(0,0,0);
                }
        }

prnbuf::prnbuf(int f)
        {
        xfd = EOF;
        isBinary = 0;

        // Check that port is supported.

        unsigned u = ((unsigned)(biosequip()) >> 14) & 0x0003;
        // u is the number of printers supported by BIOS.

        if( f >= 0 && f < u )
                {

                // Make sure printer is there.

                unsigned u = biosprint( _PRINTER_STATUS, 0, f );
                if( (u & pfSelected) == 0 )
                        return;
                xfd = f;
                }
```

```
        char* p = new char[B_size];
        if( p )
                {
                setb(p, p+B_size, 1);    // ~streambuf() will delete.
                setp(base(),ebuf());
                setg(0,0,0);
                }
        }

prnbuf::prnbuf(int f, char* buf, int len)
        {
        xfd = EOF;
        isBinary = 0;

        // Check that port is supported.

        unsigned u = ((unsigned)(biosequip()) >> 14) & 0x0003;
        if( f >= 0 && f < u )
                {

                // Make sure printer is there.

                unsigned u = biosprint( _PRINTER_STATUS, 0, f );
                if( (u & pfSelected) == 0 )
                        return;
                xfd = f;
                }

        setbuf(buf, len);
        setp(base(),ebuf());
        setg(0,0,0);
        }

prnbuf::~prnbuf()
        {
        if( xfd >= 0 )
                sync();             // flush the buffer.
        }

void prnbuf::init()
        {
        if( xfd < 0 )
                return;
        biosprint( _PRINTER_INIT, 0, xfd );
        }

prnbuf* prnbuf::attach(int f)
        {
        // See if supported by equipment list.

        unsigned u = ((unsigned)(biosequip()) >> 14) & 0x0003;
        if( f >= 0 && f < u )
                {

                // Make sure printer is there.

                unsigned u = biosprint( _PRINTER_STATUS, 0, f );
                if( (u & pfSelected) == 0 )
                        return 0;
```

```
        }
    else
        return 0;

    // If already attached to a port, flush.

    if( xfd >= 0 && sync() == EOF )
        return 0;       // do NOT un-attach if flush fails.

    xfd = f;    // assumed to be valid (IE on, not out of paper...)
    isBinary = 0;
    char *b = base();   // buffer address
    if( ! b )
        {
        b = new char[B_size];
        if( b )
            {
            setb( b, b+B_size, 1 );  // ~streambuf() will delete.
            setp(base(),base());
            setg(0,0,0);
            }
        }
    return this;
    }

int prnbuf::overflow(int c)
    {
    unsigned st;

    if( xfd < 0 )
        return EOF;

    if( unbuffered() || !base() )
        {
        if( c != EOF )
            {
            char b = c;
            st = biosprint( _PRINTER_WRITE, b, xfd );
            if( (st & (pfOutOfPaper | pfIoError | pfTimeout)) != 0 )
                return EOF;

            if( b == '\n' && !isBinary )
                {
                st = biosprint( _PRINTER_WRITE, '\r', xfd );
                if( (st & (pfOutOfPaper | pfIoError | pfTimeout)) != 0 )
                    return EOF;
                }
            }
        }
    else    // flush.
        {
        char *p = pbase();
        char *ep = pptr();
        while( p < ep )
            {
            st = biosprint( _PRINTER_WRITE, *p, xfd );
            if( (st & (pfOutOfPaper | pfIoError | pfTimeout)) != 0 )
                {
                setp( p, ebuf() );
```

```
                                return EOF;
                                }
                        if( *p == '\n' && !isBinary )
                                {
                                st = biosprint( _PRINTER_WRITE, '\r', xfd );
                                if( (st & (pfOutOfPaper | pfIoError | pfTimeout)) != 0 )
                                        return EOF;
                                }
                        p++;
                        }
                setp( base(), ebuf() );
                if( c != EOF )
                        sputc(c);
                }
        return 1;
        }

int prnbuf::underflow()
        {
        if( xfd < 0 )
                return EOF;

        return biosprint( _PRINTER_STATUS, 0, xfd );
        }

int prnbuf::sync()
        {
        if( xfd < 0 )       return EOF;
        return overflow(EOF);
        }


prnstreambase::prnstreambase()
                : buf()
        {
        ios::init(&buf);
        }

prnstreambase::prnstreambase(int f)
                : buf(f)
        {
        ios::init(&buf);
        if( buf.fd() < 0 )
                clear(ios::badbit);
        else
                clear(ios::goodbit);
        }

prnstreambase::prnstreambase(int f, char* b, int len)
                : buf(f, b, len)
        {
        ios::init(&buf);
        if( buf.fd() < 0 )
                clear(ios::badbit);
        else
                clear(ios::goodbit);
        }

prnstreambase::~prnstreambase()
```

```
        {
        }

void prnstreambase::attach(int f)
        {
        if( buf.attach(f) )
                clear(ios::goodbit);
        else
                clear(ios::badbit);
        }

void prnstreambase::setbuf(char* b, int len)
        {
        if( buf.setbuf(b, len) )
                clear(ios::goodbit);
        else
                setstate(ios::failbit);
        }

prnstream::prnstream() :
                prnstreambase(),
                ostream()
        {
        }

prnstream::prnstream(int f) :
                prnstreambase(f),
                ostream()
        {
        }

prnstream::prnstream(int f, char* b, int len) :
                prnstreambase(f, b, len),
                ostream()
        {
        }

prnstream::~prnstream()
        {
        }
```

```
// this file is pstream.h and was obtained from compuserve
// the author is pat reilly
// this file contains header information for the pstream.cpp file
// for more information on the prnstream classes, see the pstream.cpp file
// for more detailed notes

#define PrnstreamVersion        1.01

#ifndef __cplusplus
#error Must use C++ for the type prnstream.
#endif

#if !defined( __PSTREAM_H )
#define __PSTREAM_H

#if !defined( __IOSTREAM_H )
#include <iostream.h>
#endif

#define B_size          1024
// B_size is the size of the default printer buffer. This size can be
// varied depending on the size of the built-in printer buffer. If the
// expected printer buffer is large, this value can be relatively small;
// if the printer has a small (or non-existent) buffer, 1K should work
// fairly well.

const   pfTimeout =             0x0001,
                pfIoError =             0x0008,
                pfSelected =    0x0010,
                pfOutOfPaper =          0x0020,
                pfAcknowledge =         0x0040,
                pfNotBusy =             0x0080;
// These constants are the bit masks for the printer status.

_CLASSDEF(prnbuf)
_CLASSDEF(prnstreambase)
_CLASSDEF(prnstream)

// ---------------------------------------------------
// Class: prnbuf
// Derived from: streambuf
// Desc:
//          Class prnbuf is the streambuf-derived class that will buffer the
//        stream output, and do the actual writes to the printer port. Uses
//        biosprint() RTL function to do actual printing, so print errors
//        (such as out-of-paper, etc) will NOT cause Dos INT 24h (error handler)
//        to be called (this is the 'Abort, Retry, ...' message). Instead, class
//        ios state flags can be used to determine printer errors.
// Note:
//          A design decision was required on whether or not the constructors
//        should send the INIT signal to the printer to verify that a printer
//        is attached and operational. The benefit of sending the signal is that
//        the printer will be initialized when you 'open' the stream (ie any
//        pages will be ejected, defaults will be established). This is also
//        detrimental: there are many cases where you don't WANT the printer
//        settings mucked with. The first version was ambiguous on this point;
```

```
//        this version (1.01) NEVER initializes the printer in the ctor; if you
//        want the printer initialized, call the public method init(). All the
//        ctor does is send the STATUS signal to the printer to verify its
//        operational status - this does not effect printer settings.
//
// Methods:
//        prnbuf()
//                Creates a buffer that is not attached to any printer port. A buffer
//                of size B_size is allocated.
//
//        prnbuf( int f )
//                If f is a BIOS supported printer port (0=LPT1, 1=LPT2, etc) then
//                creates a buffer of size B_size and attaches it to this printer
//                port.
//
//        prnbuf( int _f, char _FAR *buf, int sz )
//                If _f is a BIOS supported printer port (0=LPT1, 1=LPT2,etc) then
//                uses the sz size buffer buf and attaches it to this printer port.
//
//        ~prnbuf()
//                Destroys the buffer, first flushing any data still remaining in
//                the buffer. If the ctors prnbuf(), prnbuf(int) were used to create
//                the buffer, or attach(int) was called and the buffer was empty,
//                then ~streambuf() will deallocate the buffer.
//
//  int fd()
//                Returns the 'file descriptor' (terminology used to conform to the
//                file streams); 0=LPT1, 1=LPT2, etc, EOF = non attached to a printer
//                port.
//
//  void init()
//                Initializes the printer (if fd() != EOF) by sending the INIT signal
//                to the printer. On most printers, this should 1) eject any pages
//                currently being 'partially' printed (on page printers like the
//                HP Laserjet/Deskjet), and 2) reset the printer settings such as
//                page size, current font, paper source, etc.
//
//  void setBinary( int state )
//   Sets/Clears the binary mode if state is Non-zero/zero. In binary
//                mode, every byte sent to the stream remains unchanged; in non-
//                binary mode (ie 'cooked' mode) every '\n' sent to the printer
//                (ie line feed) will have an additional '\r' sent by prnbuf
//                (ie carriage return).
//
//  prnbuf *attach( int f )
//   Attaches this prnbuf to a printer port. If currently attached to a
//                port, the buffer is first flushed. If an internal buffer has not
//                previously been allocated, a B_size buffer will be allocated. Sends
//                a STATUS request to verify a printer is attached.
//
//  int overflow( int ch = EOF )
//   Used internally to flush the buffer (ch = EOF) or when the buffer
//                is full and another char is written to it. Sends data to the port
//                using biosprint() RTL function. Returns EOF on error so that an
//                owning stream can set the proper ios flags.
//
//  int underflow()
//   Normally in a streambuf class, this method is used to extract chars
//                from the stream. Since printers are essentially an output stream
```

```
//                  device, this method only returns the stream status. Therefore you
//                  check an open prnbuf status by performing:
//                          status = thisbuf.sgetc();
//
//   int sync()
//     Flushes the buffer to the printer.
//
// Members:
//   xfd
//                  The attached printer port (0=LPT1, 1= LPT2, etc, EOF = none).
//
//   isBinary
//                  Flag for binary mode setting (0='cooked', <>0 = 'raw').
// ------------------------------------------------

class  _CLASSTYPE prnbuf : public streambuf
        {
        public:

        // constructors, destructor
                _Cdecl prnbuf();            // make a closed prnbuf
                _Cdecl prnbuf(int);    // make a prnbuf attached to fd
                _Cdecl prnbuf(int _f, char _FAR *, int); // same, with specified buffer
                _Cdecl ~prnbuf();

                int _Cdecl fd();       // what is the file descriptor
                void _Cdecl init();                    // Printer initialization.
                void _Cdecl setBinary( int );       // set/clear binary mode.

                prnbuf _FAR * _Cdecl attach(int);  // attach this prnbuf to port

// number.

/*
 * These perform the streambuf functions on a prnbuf
 * Get and Put pointers are kept together
 */
                virtual int _Cdecl overflow(int = EOF);
                virtual int _Cdecl underflow();
                virtual int _Cdecl sync();

        protected:
                int xfd;       // the file descriptor, EOF if closed
                short isBinary;   // flag.

        };

inline int  _Cdecl prnbuf::fd()       { return xfd; }
inline void _Cdecl prnbuf::setBinary( int st )
        {
        isBinary = st ? 1 : 0;
        }

// ------------------------------------------------
// Class: prnstreambase
// Derived from: ios (virtual)
// Desc:
//          Method that contains the methods common to all printer streams.
// Methods:
```

```
//      prnstreambase()
//    Creates a stream base that is not attached to a printer.
//
//      prnstreambase( int p )
//              Creates a stream base attached to printer port p (0=LPT1, 1=LPT2,
//              etc).
//
//      prnstreambase( int _f, char *buf, int sz )
//              Creates a stream base attached to printer port _f using a sz-sized
//              buffer whose pointer is buf.
//
//      ~prnstreambase()
//              Flushes the buffer, deallocates as necessary.
//
//      void attach( int p )
//              If currently attached to a port, flushes the buffer. Then attaches
//              to port p and allocates a buffer if one has not already been
//              created.
//
//      void init()
//              Calls prnbuf::init() to send the INIT signal to the printer.
//
// void setBinary( int flag )
//    Calls prnbuf::setBinary(flag) to set/clear the binary mode.
//
//      void setbuf(char *buf, int sz )
//    Calls prnbuf::setbuf(buf,sz) (which calls streambuf::setbuf(buf,sz))
//              To change the buffer to use buf (sz-sized). If the old buffer was
//              allocated by prnbuf, it will be deleted.
//
//      prnbuf *rdbuf()
//    Returns a pointer to the prnbuf.
//
// Members:
//      buf
//              Prnbuf stream buffer.
// ---------------------------------------------

class _CLASSTYPE prnstreambase : virtual public ios
        {
        public:
                _Cdecl prnstreambase();
                _Cdecl prnstreambase( int);
                _Cdecl prnstreambase( int _f, char _FAR *, int );
                _Cdecl ~prnstreambase();

                void   _Cdecl attach(int);
                void    _Cdecl init();
                void    _Cdecl setBinary( int );
                void   _Cdecl setbuf(char _FAR *, int);
                prnbuf _FAR * _Cdecl rdbuf();

        private:
                prnbuf buf;
        };

inline prnbuf _FAR * _Cdecl prnstreambase::rdbuf() { return &buf; }
inline void prnstreambase::init()
        {       buf.init();       }
```

```
inline void prnstreambase::setBinary( int st )
        {       buf.setBinary(st);      }

// ------------------------------------------------------------
// Class: prnstream
// Derived from: prnstreambase, ostream
// Desc:
//          Class prnstream implements a standard output stream (ostream) that
//        streams to a printer port. All of the public method of prnstreambase
//        are available, as well as the standard ostream methods, including
//        manipulators and overloaded operators.
// Methods:
//        prnstream()
//                Create a stream that is not attached to a printer port.
//
//        prnstream( int p )
//                Create a stream and attach to printer port p (0=LPT1, 1=LPT2, etc).
//
// prnstream( int _f, char *buf, int sz )
//                Create a stream and attach to printer port _f(0=LPT1, 1=LPT2, etc).
//                Use the sz-sized buffer buf for buffered output.
//
//        ~prnstream()
//                Flushes the stream and deallocated as necessary.
//
//        prnbuf *rdbuf()
//                Returns a pointer to the prnbuf stream buffer.
// ------------------------------------------------------------

class _CLASSTYPE prnstream : public prnstreambase, public ostream
        {
        public:
                _Cdecl prnstream();
                _Cdecl prnstream(int);
                _Cdecl prnstream(int _f, char _FAR *, int);
                _Cdecl ~prnstream();

                prnbuf _FAR * _Cdecl rdbuf();
        };

inline prnbuf _FAR * _Cdecl prnstream::rdbuf() { return prnstreambase::rdbuf(); }

#endif
```

```
// this file is ptest.cpp and is used in the ferrylog program

/* this file contains code to assist in determining if the printer is
on and working properly, and if not to report it to the user.  This
code uses the pstream.cpp classes.

An suggestion on improving the program:
Modify the printer check routines to make them more robust.  As it
now stands, checks are normally accomplished only when the printers are
selected on using the print options view.  Improve the code so that
checks are done for an offline or out of paper printer are done more
frequently, perhaps by changing the pstream class or the << operator.
*/

#include <stdio.h>
#include <conio.h>
#include <bios.h>
#include "consts.h"
#include <string.h>
#include "pstream.h"

#include "utilfncs.h"
extern prnstream PrinterStream;

PrinterStatusInfo PrinterStatus(void) {
        const int MaxErrLen=20;
        int status=0, return_value;
        char text[MaxErrLen];
        strncpy(text," ", MaxErrLen);  // init the string

        PrinterStatusInfo returnInfo;


        status = PrinterStream.rdbuf()->sgetc();

        if(status ==(-1)) {  // EOF has been raised as a signal that
                             // the printer is not operational
                strncpy (text, "Printer Problem",MaxErrLen);
                return_value=No;
        }
        else if (status & (pfSelected | pfNotBusy)) {
                // online and ready
                strncpy(text,"Ready.",MaxErrLen);
                return_value=Yes;
        }
        else if (status & pfTimeout){
                strncpy(text,"Device time out.",MaxErrLen);
                return_value=No;
        }
        else if (status & pfIoError){
                strncpy(text,"I/O error.",MaxErrLen);
                return_value=No;
        }
        /* else if (status & pfSelected){
                strncpy(text,"Selected.",MaxErrLen);
                return_value=Yes;
```

```
        } */
        else if (status & pfOutOfPaper){
                strncpy(text,"Out of paper.",MaxErrLen);
                return_value=No;
        }
        else if (status & pfAcknowledge){
                strncpy(text,"Acknowledge.",MaxErrLen);
                return_value=No;        // is this correct ?
        }
        /* else if (status & pfNotBusy) {
                strncpy(text,"Not busy.");
                return_value=No;
        }
        */

        else {  // unknown problem
                strncpy (text, "Unknown Problem.",MaxErrLen);
                return_value=No;
        }

        strncpy (returnInfo.PrinterStatusText, text, MaxErrLen);
        returnInfo.PrinterStatusInt=return_value;
        return (returnInfo);
}
```

```
/*-------------------------------------------------------*/
/* filename -       tinputli.cpp                         */
/*                                                       */
/* function(s)                                           */
/*                 TInputLine member functions           */
/*-------------------------------------------------------*/

/*-------------------------------------------------------*/
/*                                                       */
/*   Turbo Vision -  Version 1.0                         */
/*                                                       */
/*                                                       */
/*   Copyright (c) 1991 by Borland International         */
/*   All Rights Reserved.                                */
/*                                                       */
/*-------------------------------------------------------*/

// Modified by Erik H. Beck
// this is file TInputli.cpp and was modified from the standard TVIsion
/* Library because of a nasty bug in the library.  If the cursor in a
TInputLine is in insert mode, it is possible for the cursor to
easily overrun the bounds of the input line, causing a heap corruption and
the system to hang.  This is a problem in the library that Borland
doesn't seem interested in fixing.  A slight modification was made to
prevent the cursor from being put into the insert mode that would cause the
crash, yet allow data to still be put into the input line.  The changed
line is noted below. */

#define Uses_TKeys
#define Uses_TInputLine
#define Uses_TDrawBuffer
#define Uses_TEvent
#define Uses_opstream
#define Uses_ipstream
#include <tv.h>

#if !defined( __CTYPE_H )
#include <ctype.h>
#endif  // __CTYPE_H

#if !defined( __STRING_H )
#include <String.h>
#endif  // __STRING_H

#if !defined( __DOS_H )
#include <Dos.h>
#endif  // __DOS_H

const int CONTROL_Y = 25;

char hotKey( const char *s )
{
   char *p;

   if( (p = strchr( s, '~' )) != 0 )
      return toupper(p[1]);
   else
```

```
    return 0;
}

#define cpInputLine "\x13\x13\x14\x15"

TInputLine::TInputLine( const TRect& bounds, int aMaxLen ) :
  TView(bounds),
  data( new char[aMaxLen] ),
  maxLen( aMaxLen-1 ),
  curPos( 0 ),
  firstPos( 0 ),
  selStart( 0 ),
  selEnd( 0 )
{
  state |= sfCursorVis;
  options |= ofSelectable | ofFirstClick;
  *data = EOS;
}

TInputLine::~TInputLine()
{
  delete data;
}

Boolean TInputLine::canScroll( int delta )
{
  if( delta < 0 )
    return Boolean( firstPos > 0 );
  else
    if( delta > 0 )
      return Boolean( strlen(data) - firstPos + 2 > size.x );
    else
      return False;
}

ushort TInputLine::dataSize()
{
  return maxLen+1;
}

void TInputLine::draw()
{
  int l, r;
  TDrawBuffer b;
// The change below causes the input field to blink when it is selected
// This blinking occurs because the getcolor routine is trying to access
// an inappropriate color location, and is displaying the default
// warning message of a red and white blinking state.
// this change was made to make the field easier to locate on a monochrome
// screen

//   uchar color = (state & sfFocused) ? getColor( 2 ) : getColor( 1 );
// the above is the original line, below is the changed line
  uchar color = (state & sfFocused) ? getColor( 99 ) : getColor( 1 );

  b.moveChar( 0, ' ', color, size.x );
  char buf[256];
  strncpy( buf, data+firstPos, size.x - 2 );
  buf[size.x - 2 ] = EOS;
```

```
    b.moveStr( 1, buf, color );

    if( canScroll(1) )
        b.moveChar( size.x-1, rightArrow, getColor(4), 1 );
    if( (state & sfSelected) != 0 )
      {
            if( canScroll(-1) )
            b.moveChar( 0, leftArrow, getColor(4), 1 );
            l = selStart - firstPos;
        r = selEnd - firstPos;
        l = max( 0, l );
        r = min( size.x - 2, r );
        if (l < r)
            b.moveChar( l+1, 0, getColor(3), r - l );
      }
    writeLine( 0, 0, size.x, size.y, b );
    setCursor( curPos-firstPos+1, 0);
}

void TInputLine::getData( void *rec )
{
    memcpy( rec, data, dataSize() );
}

TPalette& TInputLine::getPalette() const
{
    static TPalette palette( cpInputLine, sizeof( cpInputLine )-1 );
    return palette;
}

int TInputLine::mouseDelta( TEvent& event )
{
    TPoint mouse = makeLocal( event.mouse.where );

    if( mouse.x <= 0 )
        return -1;
    else
        if( mouse.x >= size.x - 1 )
            return 1;
        else
            return 0;
}

int TInputLine::mousePos( TEvent& event )
{
    TPoint mouse = makeLocal( event.mouse.where );
    mouse.x = max( mouse.x, 1 );
    int pos = mouse.x + firstPos - 1;
    pos = max( pos, 0 );
    pos = min( pos, strlen(data) );
    return pos;
}

void  TInputLine::deleteSelect()
{
    if( selStart < selEnd )
      {
        strcpy( data+selStart, data+selEnd );
            curPos = selStart;
```

```
    }
}

void  TInputLine::handleEvent( TEvent& event )
{
   TView::handleEvent(event);

   int delta, anchor, i;
   if( (state & sfSelected) != 0 )
      switch( event.what )
       {
        case  evMouseDown:
                  if( canScroll(delta = mouseDelta(event)) )
           do {
                           if( canScroll(delta) )
              {
                firstPos += delta;
               drawView();
              }
               } while( mouseEvent( event, evMouseAuto ) );
           else if (event.mouse.doubleClick)
              selectAll(True);
          else
           {
             anchor =  mousePos(event);
                    do {
              if( event.what == evMouseAuto &&
                              canScroll( delta = mouseDelta(event) )
            )
                firstPos += delta;
              curPos = mousePos(event);
             if( curPos < anchor )
             {
               selStart = curPos;
               selEnd = anchor;
             }
            else
             {
                              selStart = anchor;
               selEnd = curPos;
                             }
             drawView();
               } while (mouseEvent(event, evMouseMove | evMouseAuto));
          }
          clearEvent(event);
         break;
        case  evKeyDown:
           switch( ctrlToArrow(event.keyDown.keyCode) )
           {
             case kbLeft:
              if( curPos > 0 )
                              curPos--;
             break;
                    case kbRight:
              if( curPos < strlen(data) )
                curPos++;
             break;
            case kbHome:
              curPos =  0;
```

```
    break;
  case kbEnd:
     curPos = strlen(data);
    break;
  case kbBack:
                    if( curPos > 0 )
     {
                        strcpy( data+curPos-1, data+curPos );
       curPos--;
       if( firstPos > 0 )
         firstPos--;
     }
    break;
  case kbDel:
     if( selStart == selEnd )
       if( curPos < strlen(data) )
       {
         selStart = curPos;
                              selEnd = curPos + 1;
       }
                    deleteSelect();
    break;
          /* case kbIns:
       setState(sfCursorIns, Boolean(!(state & sfCursorIns)));
                    break;*/
                    // removed to disable bug described at top of file
             default:
                    if( event.keyDown.charScan.charCode >= ' ' )
                      {
                       if( (state & sfCursorIns) != 0 )
                              strcpy( data + curPos, data + curPos + 1 );
                       else
                              deleteSelect();
       if( strlen(data) < maxLen )
                              {
         if( firstPos > curPos )
           firstPos = curPos;
          memmove( data + curPos + 1, data + curPos,
              strlen(data+curPos)+1 );
         data[curPos++] =
             event.keyDown.charScan.charCode;
       }
     }
       else if( event.keyDown.charScan.charCode == CONTROL_Y)
     {
       *data = EOS;
       curPos = 0;
                      }
    else
       return;
}
  selStart = 0;
  selEnd = 0;
  if( firstPos > curPos )
     firstPos = curPos;
  i = curPos - size.x + 3;
  if( firstPos < i )
     firstPos = i;
  drawView();
```

```
            clearEvent( event );
            break;
        }
}

void TInputLine::selectAll( Boolean enable )
{
  selStart = 0;
  if( enable )
    curPos = selEnd = strlen(data);
  else
    curPos = selEnd = 0;
  firstPos = max( 0, curPos-size.x+3 );
  drawView();
}

void TInputLine::setData( void *rec )
{
  memcpy( data, rec, dataSize()-1 );
  data[dataSize()-1] = EOS;
  selectAll( True );
}

void TInputLine::setState( ushort aState, Boolean enable )
{
  TView::setState( aState, enable );
  if( aState == sfSelected ||
    ( aState == sfActive && (state & sfSelected) != 0 )
  )
    selectAll( enable );
}

void TInputLine::write( opstream& os )
{
  TView::write( os );
  os << maxLen << curPos << firstPos
    << selStart << selEnd;
  os.writeString( data);
}

void *TInputLine::read( ipstream& is )
{
  TView::read( is );
  is >> maxLen >> curPos >> firstPos
    >> selStart >> selEnd;
  data = new char[maxLen + 1];
  is.readString(data, maxLen+1);
  state |= sfCursorVis;
  options |= ofSelectable | ofFirstClick;
  return this;
}

TStreamable *TInputLine::build()
{
  return new TInputLine( streamableInit );
}
TInputLine::TInputLine( StreamableInit ) : TView( streamableInit )
{
}
```

```
/*This file is tmdis.cpp and is used to build the ferrylog program.
This file contains code to control the recording of a time different from
the system time.  This facility uses a pair of buttons to allow the user
to adjust the system time up or down from the current time as needed.
*/

// this subroutine was designed to interface with the TimeGrabber object
// that appears in another source file


#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <stdio.h>
#include "tmdis.h"



char * TimeDisplay::FormatVert(void) {
        // puts the integers back into character form

        char temp [6];
        char *return_value;
        temp[5]='\0';
        temp[0]=(Hours / 10)+48;
        temp[1]=(Hours % 10)+48;
        temp[2]=':' ; // put colon back in
        temp[3]=(Mins / 10)+48;
        temp[4]=(Mins % 10)+48;
        strncpy(return_value, temp, 6);
        return return_value;
}

void TimeDisplay::TimeDisplaySet ( time_t default_time) {

        // get initial time (current system time)
        //Convert time_t into TimeDisplay data elements

        char longtime[26];
        char shorttime[6];
        int intTime=0;

        strcpy (longtime, ctime (&default_time) );
        sscanf(&longtime[11], "%5s", shorttime);
        // now we need to remove the colon
        shorttime[2]=shorttime[3];
        shorttime[3]=shorttime[4];
        shorttime[4]=shorttime[5];
        // convert to integer
        intTime=atoi(shorttime);
        Hours=(intTime/100);
        Mins=(intTime %100);

}
```

```
void TimeDisplay::Inc (void) {
        if (Mins !=59) Mins++;
        else {
                Mins=0;
                if (Hours!=23) Hours++;
                else {
                        Hours=0;
                        Mins=0;
                }
        }
        // inputtime
        return;
}


// decrement
void TimeDisplay::Dec (void) {

        if (Mins !=0) Mins--;
        else {
                Mins=59;
                if (Hours!=0) Hours--;
                else {
                        Hours=23;
                        Mins=59;
                }
        }
        return;
}
```

```
// this file is tmdis.h
// this file is used in the ferrylog program, and
// contains header information for the time display and alteration
// class that interacts with the TimeGrabber class.

#include <time.h>
class TimeDisplay{
// used only by time grabber class below
public:
        int Mins;
        int Hours;
        TimeDisplay (void) {Mins=0; Hours=0;}
        void TimeDisplaySet (time_t default_time);
        void Inc (void);
        void Dec (void);
        char * FormatVert(void);
};
```

UTILFNCS.H

```
// this file is utilfncs.h

/* It is a header file for utility functions  */


#if!defined(__UTILFNCS_H)
#define __UTILFNCS_H
#include "pstream.h"
int mycopy (char * infilename, char *outfilename);
// structure for printer status
class PrinterStatusInfo {
public:
        char *PrinterStatusText;
        int PrinterStatusInt;
        //void PrinterError (char *errorMsg);
};

// printer status header
PrinterStatusInfo PrinterStatus(void);
// sound function
void MakeASound(int );
const char near * VesselNameLookup (int  vessel_cmd);
const char near * RouteNameLookup (int  route_cmd_number);


#endif
```