ISSN 0280–5316
ISRN LUTFD2/TFRT--5473--SE

Flight Operations Support System for X11/MOTIF

Jimmy C. M. Lu

Department of Automatic Control
Lund Institute of Technology

May 1993

Department of Automatic Control **Lund Institute of Technology** P.O. Box 118 S-221 00 Lund Sweden	*Document name* MASTER THESIS *Date of issue* May 1993 *Document Number* ISRN LUTFD2/TFRT--5473--SE
Author(s) Jimmy C. M. Lu	*Supervisor* Rolf Johansson and Jonas Fredenholm *Sponsoring organisation*

Title and subtitle

Flight Operations Support System for X11/MOTIF

Abstract

This master thesis is based on a project of Flight Operation Support Systems, where all navigational aids and airport facilities are represented as symbols on the screen. Information about each object can be retrieved by a click from the mouse attached to the computer, or by using a searching criteria.

When creating databases with tens of thousands of elements, it is important to consider the time involved for searching a particular element. A way to shorten the search time is to have a coding method that will ease the searching. Hash coding has been chosen to achieve an efficient method to fulfill this purpose.

In order for users to adapt to the software within a minimum range of time, a graphical user interface has been chosen. Navigational objects are presented as cartographic symbols on the screen, where information can be retrieved by simply using the mouse or track ball to point and select. Several functions are available from a menu where functions to be activated are tied to toggleButtons attached to the menu. To activate the functions, simply select the toggleButton, of which the function is attached to, by either using a pointing device or the keyboard. More complex functions are supported by having "popup" menus where multiple choices are presented either as toggleButtons or texField for input.

Optimization of flight routes to gain minimal flight operational costs, or to find the shortest airway between two points, is not an easy task due to the intricacy of the airway network. Consequently the main concern of this project is focused on finding an appropriate and reliable method for optimization of flight routes. The method used is a modification of *dynamic programming.*

The software package is implemented under the programming language C^{TM} with support of the graphical software development package OSF/MOTIF^{TM} layered on the top of the X Window System^{TM} using the operating system UNIX^{TM}.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 58	*Recipient's notes*
Security classification		

The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Fax +46 46 110019, Telex: 33248 lubbis lund.

Flight Operations Support System For X11/MOTIF

Table Of Content

Acknowledgment

A very special thanks:

To Rolf Johansson, associate professor in the Department of Automatic Control, for his supervision of this thesis.

To Jonas Fredenholm, Director of Operations Engineering, of Flygprestanda AB, for his invaluable guidance, advice and expertise. This project would not have been possible without him.

To my dearest wife, Isabel, for all her love and support, not forgetting all the delicious meals during my stint at Flygprestanda AB.

1. Task

1.1. Background

Flygprestanda AB is a consultation company that provides aircraft analysis for airlines and air cargo companies. Its services includes the improvement of fuel burn figures, maximization of takeoff and landing weights, ensuring of safe takeoff and landing weights on runways in inclement weather.

At Flygprestanda, when information about a certain navigational aid object is retrieved, a text string is displayed on screen with the information needed. The precise geographical situation of the navaid is normally uncertain to the operator, unless the effort is taken to find out the navaid position on a map. Having a graphical interface where objects searched are presented on the screen with their relating objects and the accurate geographical situation, will increase the lucidity of the overall information retrieving operation.

Flight planning is normally done in rather primitive way, where ruler and pencil are the only tools used together with maps containing pre-defined airways and navigational aids. The method of designing routes in this way is not only time consuming, it is also not accurate in terms of getting the shortest route possible.

One of the reasons why an automated route planning system has not been implemented at Flygprestanda is due to the high cost of having a computer that has adequate computing power and memory to maintain the airway network in order to handle the route optimization. The micro computers of today actually have the same or better computing power than a mini computer of yesterday, the primary storage is also more easily available at a much lower cost compared to prises a couple of years ago.

Having a fully computerized flight planning system where the route optimization involving the entire globe that just takes a couple of minutes, and to have the results presented on screen (to reassure the accuracy of the designed route to be the shortest or cheapest concerning the optimize criteria); not only shorten the time spent as compared to the method mentioned above, it is also a user friendly system.

1.2. Specification

Title: Flight Operation Support System

The thesis will constitute of two parts:

- A software package for showing and manipulating a flight facilities database in a X11 environment.

- Documentation.

1. Software description

The software shall be written in C and uses X11 Xtoolkit.

The following function shall be implemented:

a) A function / widget package showing airfields, navigation aids, airways and geographical areas of importance for the flight.
The widget shall be able to:
- Showing all objects as cartographic symbols.
- Selecting all objects by mouse.
- Showing all attributes of an object after selection.
- Showing objects after a selected search command, that is searching for all objects matching a request. Example: show all open airports within a radius of 50 nautical miles.
- searching for the shortest way between two points in the airway network.
- Zoom the shown picture.
- Highlighting selected objects.
- Making an interface for calculating time and fuel for an actual aircraft flying a specified route. (Flygprestanda has functions for retrieving the aircraft specific data).

b) A widget showing the airport and the flight procedures used for takeoff and landing at the airport.
Geographical information and procedure coding supplied by Flygprestanda AB.

2. Documentation.

The documentation shall be written in English, and contain a description of algorithms used in the software package, as well as a description of the widget function and relationship.

2. Databases

For current software implementation, there are seven different databases, to be interpreted and managed individually. When hash coding is used to generate databases, it will improve the efficiency of element searching tremendously.

Flygprestanda provides the tools for generating hash tables, searching functions and database handler.

2.1. Description

The databases to be interpreted are all given as ASCII texts where information like geographical position, magnetic variation, elevation, identity and others are listed.

A separate database interpreter has been implemented where information of one object is gathered in a record. A hash table is generated for each one of the navaid database for quick access.

2.2. Object Symbols

When using a map for flight operational use, different type of navigational aid will be found as cartographic symbol on the map. It is then natural to use the same set of symbols to represent the objects when implementing the software that handles these objects, to provide users a familiar environment. The following are simple descriptions of different type of navigational aids:

1.	Airport.		○
2.	Runway:	The field where aircraft take off and land.	▨
3.	Way point:	A geographical point, used as an airway connection.	△
4.	VOR:	**V**ery high frequency **O**minidirectional **R**adio range. Part of the **VHF** database.	⬡
5.	DME:	**D**istance **M**easuring **E**quipment. Part of the **VHF** database.	□
6.	NDB:	**N**on **D**irectional **B**eacon. A radio signal that is being transmitted in all directions.	⁘
7.	Airways:	Made up of a series of route segments, each linking two way points.	

2.3. Object Description

2.3.1. General Information

There are a few terms that are frequently used below, here follows a short explanation:

- ICAO, International Civil Aviation Organization. An UN organization.
- ICAO area code, a two letter code that specifies a specific area. One of the standard in aviation field.
- ICAO airport code, a four letter code unique to every airport. The first two letters are normally the same as the ICAO area code.
- IATA, International Air Transport Association.
- Navaid, navigational aid.
- latitude, specifies the angular distance measured in degree north or south from the equator. 0° at equator, -90° at south pole and 90° at north pole.
- longitude, specifies the angular distance east or west on the earth's surface. Measured as an arc of the equator between the meridian passing through Greenwich in England. From -180° west of Greenwich to 180° east of Greenwich.
- nautical mile, a unit of length based on the length of one minute of arc of a great circle. One NM is equvalent to 1852 m, and is normally used for distance measuring in air traffic applications.
- second, one 60th of a minute.

Generally the databases are updated once every 28 days, where changes for the navaid objects are specified, and the validity date of change.

2.3.2. Airport

Airport database contains the following information:

- name of the airport, the actual name of the airport.
- ICAO area code, two letter code for the area where airport is situated.
- airport identity, ICAO code, a four letter code unique for the airport.
- geographical position of the airport in latitude and longitude.
- transition altitude in feet.
- magnetic variation at the airport.
- validity of the airport data.
- date of validity.

One hash table, with the airport ICAO code as search key, has been generated from the database file.

2.3.3. Runway

Runway database contains the following information:

- name of the runway.
- ICAO code of the airport that the runway belongs to.
- ICAO area code of the airport that the runway belongs to.
- geographical position, in latitude and longitude, of the threshold point of the runway. Threshold point is a position on the runway where aircraft have to pass by 50 ft above the ground level during landing or take off.
- the height, in feet above mean sea level, of the threshold point of the runway.
- the width of the runway, in feet.
- magnetic variation of where runway is situated.
- validity of the runway data.
- date of validity.

One hash table, with the airport ICAO code as search key has been generated from the runway database file.

2.3.4. Waypoint

A waypoint is a predetermined and accurately known geographical position forming start or end of route segment. Waypoint database contains the following information:

- ICAO area code for the waypint.
- the waypoint identifier. Unique with combination of the ICAO area code.
- name of the waypoint.
- geographical situation of the waypoint, in latitude and longitude.
- waypoint description. *ENRT* for enroute and airport ICAO code for waypoint belonging to a specific airport.
- ICAO area code for the airport that the waypoint belongs to.
- different classes of waypoint.
- validity of the waypoint data.
- date of validity.

One hash table, with the waypoint identity as search key, has been generated from the waypoint database.

2.3.5. VHF

Very High Frequency database, contains information for two type of navigational aids:

- VOR, Very high frequency Omnidirectional Radio range, transmits course guidance signals and a Morse code identification in the VHF range between 108.00 and 117.95 MHz. An aircraft can read from panel instrument its bearing from station.

- DME, Distance Measuring Equipment, provides direct readouts of distance and a Morse code identification from a selected DME. DME operates on frequencies between 962 and 1213 MHz in the UHF spectrum.

The VHF database contains the following information:

- VOR identifier.
- DME identifier.
- ICAO airport identity that the VHF belongs to.
- ICAO area code of the airport.
- different classes of VHF navaid.
- geographical situation of the VOR or DME, in latitude and longitude.
- the height above sea level, in feet, where the VHF is situated.
- the magnetic variation of where the VHF is situated.
- frequency of the VHF in MHz.
- validity of the VHF data.
- date of validity.

One hash table each for VOR and DME, with the VHF identity as search key, has been generated from the VHF database.

2.3.6. NDB

NDB, the Non Directional radio Beacon, transmits course guidance signals and a Morse code identification in the low frequency range between 190 and 535 kHz. Airborne equipment shows bearing of aircraft from station. The NDB is the most common radio navaid.

The NDB database contains the following information:

- NDB identifier.
- ICAO area code of where the NDB is situated. The identifier and the area code forms an unique identifier globally.
- geographical position of the NDB, in latitude and longitude.
- transmitting frequency of the NDB in kHz.
- magnetic variation of where the NDB is situated.
- different classes of NDB.
- validity of the NDB data.
- date of validity.

One hash table, with the NDB identity as search key, has been generated from the NDB database.

2.3.7. Airway

Airway is made up of a series of route segments each linking two waypoints. Each airway has the form of a corridor, with rectangular cross section well above earth, typically 10 nautical mile wide with centreline defined by point-source radio navaids.

The airway database contains the following information:

- airway identifier.
- three letter area code of the airway. For instance, CAN stands for Canada and AFR stands for the continent Africa.
- type of the connecting navaid.
- identifier of the connecting navaid.
- ICAO area code of the connecting navaid.
- description of the connecting navaid.
- course to be taken in order to reach the next connecting navaid.
- distance, in nautical mile, between the present connecting navaid and the next connecting navaid.
- sequence number of the airway segment.
- validity of the airway segment data.
- date of validity.

One hash table, with the airway identity as search key, has been generated from the airway database.

2.4. Summary

The number of databases is actually larger than mentioned above, due to limited time and increased complexity of the problems caused by databases remained, the databases, except the above mentioned, are left for later integration.

The databases used for this project has been interpreted and managed by the designer of this software package after consulting the responsible sources in Flygprestanda AB.

References

Bill Gunston, 1986, "JANE'S AEROSPACE DICTIONARY New Edition". Jane's Publishing Company Limited. ISBN: 0 7106 0365 7

ARINC 424-7 specification.

Flygprestanda AB archive.

3. Software Developing Environment

3.1. Description

The programming language **C** is one of the more popular and accepted programming language available, which is one of the reasons why **C** has been chosen for this project. The standard software development language under operating system **UNIX** is **C** makes the choice natural.

To develop an object oriented program using standard **C** is not an easy task, especially the graphical interfaces and windowing objects. That's why the software developer are directly dependent on certain software packages for graphics primitives and window management.

3.2. The Software Hierarchy

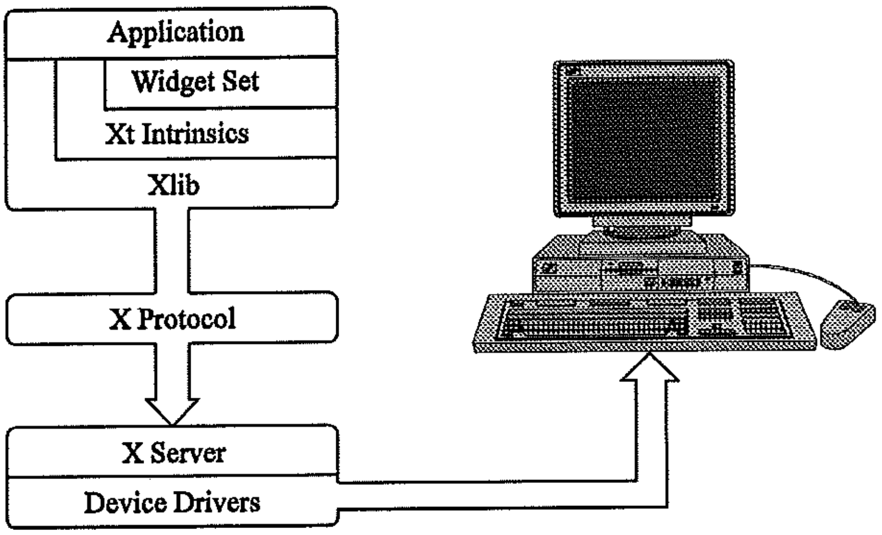


Fig. 3-1. The software architecture of Xt Intrinsic based applications.

The **X Window System** (or simply **X**) is a hardware indenpendent and operating system independent windowing system developed jointly by **MIT** and Digital Equipment Corporation.

Software development under **X** has to follow The *X protocol* to ensure compatibility of the software and hardware requests. **Xlib** is a low level **C** language interface that provides full access to the capabilities of the *X protocol.*

X Toolkit Intrinsics (Xt) is built upon **Xlib**. The purpose of **Xt** is to provide an object oriented layer that supports the user interface abstract called a widget. A *widget* is a reusable, configurable piece of code that operates independently of the application except through prearranged interactions.

A *widget set* is a collection of widgets that provides commonly used user interface components tied together with a consistent appearance and user interface.

3.3. X Toolkit Intrinsics

It is difficult to build applications that have a graphical user interface using a low level programming language such as **Xlib**. To simplify development, each of the user-interface elements of a graphical application should ideally be available ready-made. The purpose of the **X Toolkit** is to provide such a simplified approach to graphical user-interface programming.

The **X Toolkit** is a library package layered on top of the **X Window System** and provides the base functionality necessary to build a wide variety of application environments. It is fully extensible and supportive of the independent of new or extended components. This is accomplished by defining interfaces that mask implementation details from both applications and common component implementators.

The **X Toolkit** provides tools that simplify the design of application user interfaces in the **X Window System** programming environment. It consists of:

- A set of Intrinsic mechanisms for building *widgets*.
- An architectural model for constructing and composing *widgets*.
- A consistent interface (*widget set*) for programming.

The Intrinsics mechanisms are intended for the *widget* programmer. The architectural model lets the *widget* programmer design new *widgets* by using the Intrinsics or by combining other *widgets*. The application-interface layers built on top of the **X Toolkit** include a coordinated set of *widgets* and composition policies.

3.4. Widget

The fundamental data type of the **X Toolkit** is the *widget*, which is a short form for 'windowed gadget'. A widget is dynamically allocated and contains state information. Every widget belongs to one class, and each class has a structure that is statically allocated and initialized and contains operations for that class. Widgets are used either individually or in combination to make the creation of complex applications easier and faster.

A widget can be thought, to the application programmer, as a black box with associated input/output relations. Much of the input/output of a widget is customizable by the users. An instance is a specific widget object as opposed to a general widget class. A widget instance is composed of two parts:

- A data structure that contains instance-specific values.
- A class structure that contains information that is applicable to all widgets of that class.

A widget class is the procedures and data that is associated with all widgets belonging to that class. These procedures and data can be inherited by subclasses.

To get consistent 'look and feel' capabilities, it is customary to use a *widget set* along with the **Xt**.

3.5. The Motif Widget Set

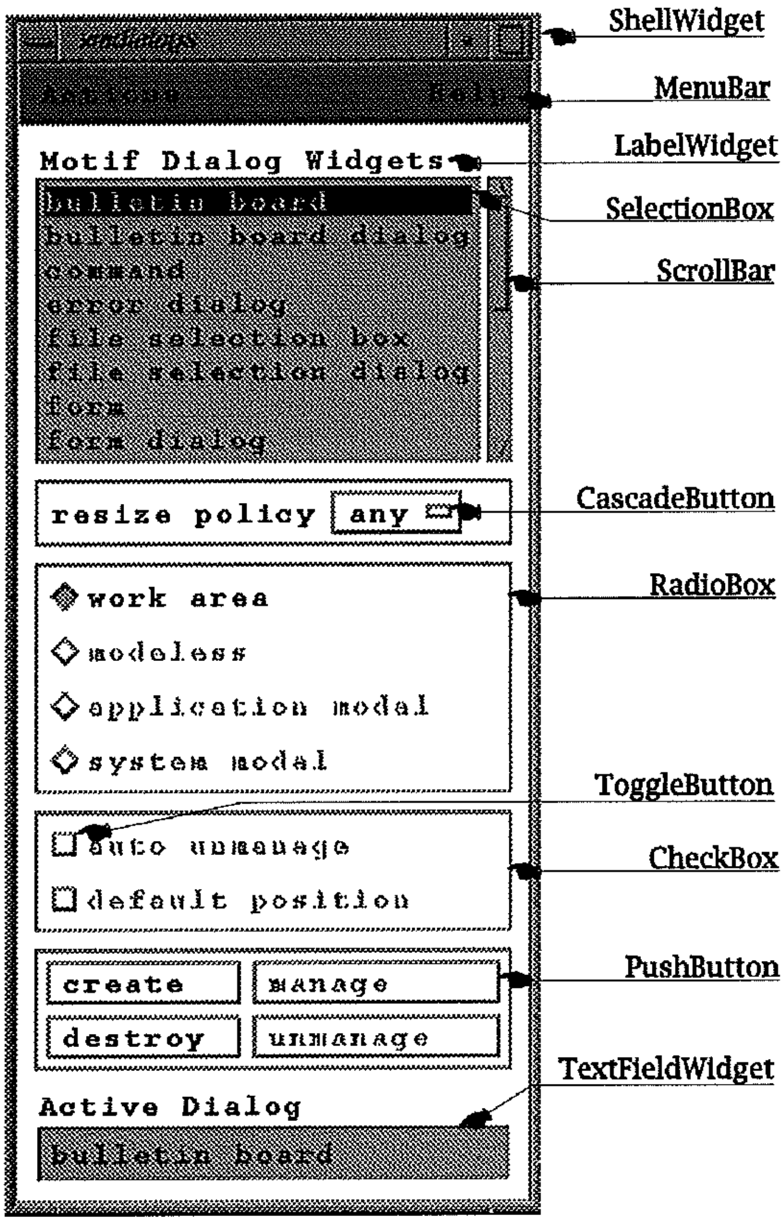


Fig. 3-2 A sample of motif widget set.

A *widget set* is a library of pre-built user-interface components. One of the *widget set* available commercially is the **Motif** system by **OSF**. **Motif** provides the application a wide range of menus, scroll bars, command buttons, dialog boxes, and a wide variety of composite widgets. **Motif** has a distinctive appearance, using shadowed outline to simulate a 3-D appearance, **Motif** has also conventions about the use of its widgets that lead to a consistent look among all applications using **Motif**. The **Motif** widgets also support some advanced features for internationalization such as language-independent "compound string", keyboard traversal and mnemonic key equivalent for invoking menu commands.

3.6. Event-Driven Programming

Programming a graphically-based window system is fundamentally different from standard procedural programming. In traditional character-based interfaces, once the application starts, it is always in control. It only knows what kind of input it will allow, and may define exclusive modes to limit that input.

In a window system, multiple graphic application may be running simultaneously, in addition to the keyboard, the user can use the mouse pointer to select data, click on buttons or scroll bars, or change the keyboard focus from one application to another. The user can suddenly switch from the keyboard to the mouse, from one application area to another. Furthermore, as the user moves and resizes windows on the screen, application windows may be obscured or redisplayed. The application must be prepared to respond to any one of many different events at any time.

An X event is a data structure sent by the server that describes something that just happened that may be of interest to the application. There are two major categories of events: user input and system side effects. For instance, the user clicking a mouse button generates an event; a window being moved on the screen also generates events. It is the server's job to distribute events to the various windows on the screen.

Event-driven window programming reduces modes to a minimum. The application simply performs some setup and then goes into a loop from which application functions may be invoked in any order as events arrive.

3.7. Widget Methods

The active part of a **Motif** application is an event sensitive endless loop where events are dispatched to the various widgets. For instance, a click on the mouse button in a pushButton widget will invoke an event that subsequently will change the appearance of the pushButton and call its Activate Callback.

Some events have a special status, and the widget function that are called as a consequence of these events are called the widget methods. Some of the most important widget methods are described as follows:

- **initialize**: Called when the widget is created by the application. Initializes private state variables in accordance with resource values.
- **expose**: Called when part of or the entire widget gets exposed after having been obscured. Restores the appearance of the widget.
- **resize**: Called when a widget is being resized. Recalculates the widget's graphical appearance based on the new size.
- **set_values**: Called when a widget resource is set by the application. Recalculates private state variables based on the new resource values.
- **destroy**: Called when the widget is to be destroyed by the application. Free dynamically allocated memory blocks.

In addition to these pre-defined methods, a widget can define action routines which will be called when following a specified event sequence.

References

Adrian Nye, 1991. "Xlib Programming Manual". O'Reilly & Associates, Inc.
ISBN: 0-937175-11-0

Adrian Nye and Tim O'Reilly, 1992. "X Toolkit Intrinsics, Programming Manual".
O'Reilly & Associates, Inc.
ISBN: 0-937175-62-5

Open Soft Foundation, 1991. "OSF/Motif Programmer's Guide". Prentice Hall.
ISBN: 0-13-640673-4

4. Data Structure

The time taken to retrieve information of one single object from a database is normally not noticable to the user, but when the number of objects needed to be obtained increases, the time of information retrieval becomes critical. Especially when data has to be interpreted and then presented as symbols on the screen. This is why, it is important to have a data structure that contains adequate information to accomplish the operations requested.

4.1. Description

As information accessed from databases will be interpreted and managed, it is an advantage to have a pre-loaded part of information that can be retrieved easily and quickly. During the initialization of the software, a network structure is being built up, to contain information needed, to ensure computation speed.

4.2. Data Structure For Navaid

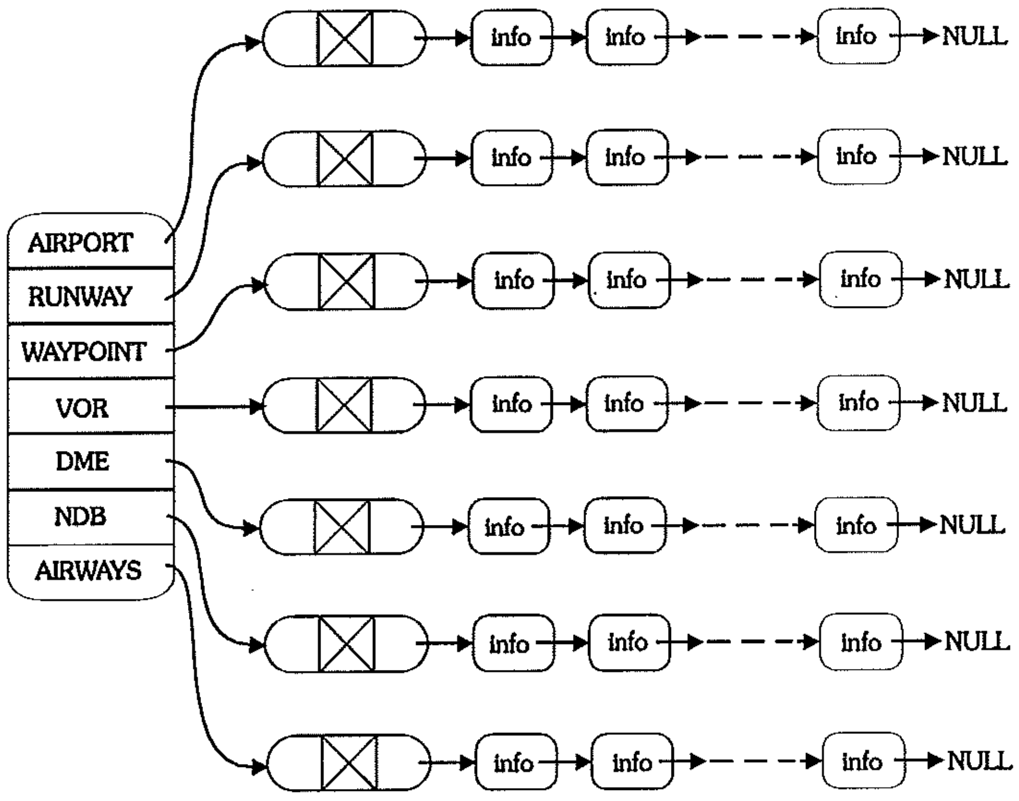


Fig. 4-1 Data structure when initialized.

The navigational aids are the essential objects for building up the airway network. The navaid databases contain, for each object, position, identity, area, and some other information, described in chapter 2.

The position of each object will be maintained in latitude and longitude, in minutes. A coordinate transformation will take place when the object is to be displayed on screen.

The following steps will occur during initialization of the software:

1. An array of pointers will be created, where each type of navigational aid is presented as a head of a linked list.

2. A linked list will be created for each type of navigational aid, where information such as geographical situation and identity for each object are stored in records that form the linked list elements.

3. During creation of the linked list for the navigational aid, a hash table will be generated to speed up the searching of navaid objects when building up the airway network.

4.3. Airway Network

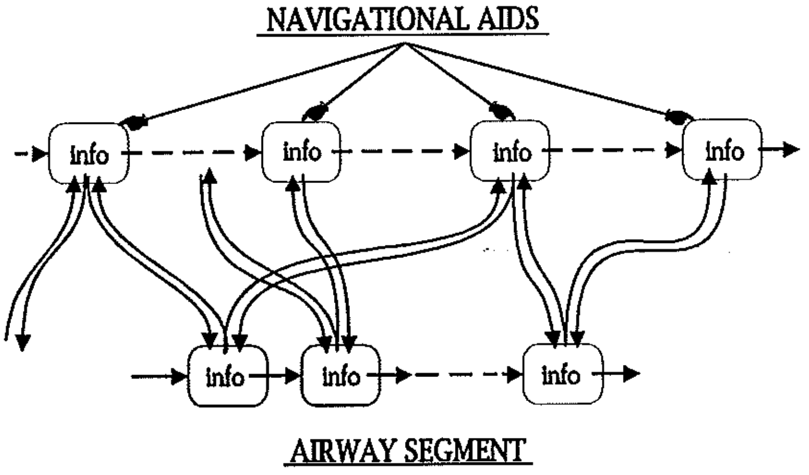


Fig. 4-2 Structure of airway network

The airway database is arranged in sequences, where each sequence has a starting segment and an ending segment. The airway segments contains information such as, distance and magnetic course and the identity for the connecting navaid and its type. The possible type of connecting nodes are, way point, VHF, and NDB. The ending airway segment has zero as distance and magnetic course to the connecting navaid.

Initialization of airway network is described in the following steps:

1. Search for navaid object, stored in the linked list described above, as specified in the airway segment.

2. From airway element issue a pointer to the specified navaid object found in step 1.
   If the airway element is **not** the beginning of the airway sequence, from airway element issue a pointer to the navaid specified in the previous airway element.

3. From navaid element issue a pointer to the airway segment post.
   If the airway element is **not** the beginning of the airway sequence; From navaid element specified in the previous airway segment, issue a pointer to the airway segment post.

4.4. Summary

The data structure and the network structure occupy a large amount of primary storage, since the entire network has to be "loaded" in the primary storage to minimize the time taken for displaying and route optimization. The amount of memory used for both the data structure and the airway network is approximately 20 Mb, which can be reduced by limiting the number of posts in the list elements, or replace all the float values with integer.

Initially the start up time was as long as 6 minutes, by using the hash tables created during the building up of navaid data structure, for initiating the network structure, the time was eventually reduced to less than 90 seconds.

5. Route Optimize

5.1. Description

The most essential task of this project is to find the shortest way between two points. It is time to look for an appropriate method for route optimization, since the airway network has been build up.

5.2. Method Used For Optimization

The method used for route optimization is a modification of *dynamic programming*, where searching starts from the point of departure, and search all possible ways out until the shortest way is found. A recursive technique has been applied for implementing the optimization routine.

When the route-optimizing routine is being called, the recursion will determine whether the destination has been reached. When the destination node is found, the recursion will return value zero. The sum of the returned value and airway segment distance will be the accumulated distance for the node where the recursion started, using the airway connection that the recursion carried out, to the destination node. If the destination node has not been reached, the recursion will carry on except when:

- the node determined is not within the search window specified. Return MAX value.
- the node discovered is the last connecting node in an airway sequence, and there are no other airway connections. Return MAX value.
- the airway segment that is in turn, has an accumulated distance that has been determined and finalized. Return the determined value.
- the node to be determined is the departure node. Return MAX value.
- the node to be determined has been visited in an earlier stage of recursion. This contributes to local loop condition. Return the value -1 to indicate that the actual value has not been determined.

To determine the shortest route to the destination node, just select the shortest accumulated distance among all airway connections. If all the airway connections have returned distance that are accurate and certain, then the selected distance automatically is certainly the shortest way to the destination node from where the recursion started. The airway segment can be closed and do not need to be determined again, even in another branch of recursion.

Due to the complexity of the airway network, all distances will not be accurately set the first time. For instance, in a loop, the distance determined is highly dependent on which sequence the recursion has carried out from a certain node, since the airway connections are numbered, the sequence of searching is always the same. This gives the propagated distance a great uncertainty that the result will not be usable.

To ensure the accuracy of the result, the search routine has to apply on the airway network again, until there are no longer changes in the airway network.

5.3. Algorithm

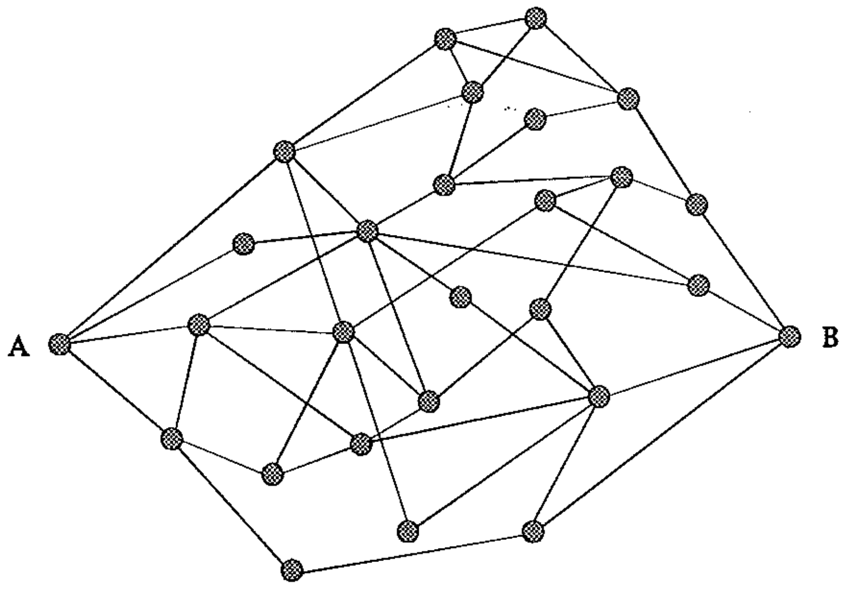


Fig. 5-1 A simplified airway network.

A recursive technique has been used for implementation of the route optimizing function. A simplified description of the algorithm is as follows:

```
SearchForTheShortestWay
    if  Destination Not Reached
        if  Not Within The Search Area
            return Maximal distance;
        for all ways out from node
                SearchForTheShortestWay
        Select the shortest way among the ways out;
        return shortest way selected;
    else  Destination Reached
        return 0;
```

5.4. Summary

The complexity of the above mentioned method was not determined due to the number of iteration needed was not certain. There might be a mathematical relation between the number of iteration and the network topology. Though the complexity is not exponential, since in every iteration, every airway segment will only be visited twice at the most, and the number of changes in the airway network decreases rather dramatically for every iteration.

When the method applies for two nodes, using the entire network structure (100 000 airway connections), on a 33Mhz 486 PC, the average system time taken was around 180 seconds, i.e. 3 minutes.

References

For further information on *Dynamic Programming* can be found in:

R. Bellman & S. Dreyfus. NJ, 1962, "Applied Dynamic Programming". Princeton University Press, Princeton.

Thomas H. Cormen, Charles E. Leiserson and Ronald L. Riverst, 1991, "Introdution to Algorithms". The MIT press and McGraw-Hill Bool Company. ISBN: 0-262-03141-8 (MIT press). ISBN: 0-07-013143-0 (McGaw-Hill).

6. Miscellaneous

6.1. Description

To accomplish this project requires more than the search algorithm and the data structure described above. There are many other routines implemented to facilitate the handling of graphics, object selection, displaying information of a selected object, displaying a selected area and objects within the area, just to mention a few.

In this chapter, some of the routines implemented will be introduced and a further explanation of their tasks and purposes will be given.

6.2. Coordinate Transformations

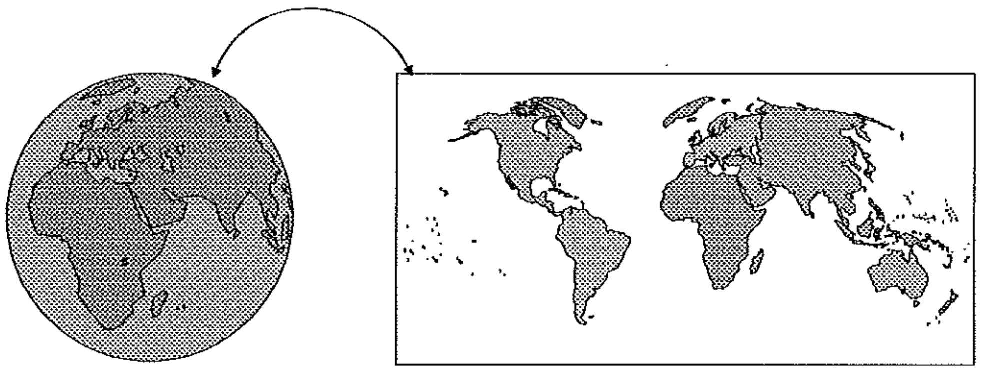


Fig. 6-1 Coordinate transformation from spherical coordinate system to Cartesian coordinate system and vice versa.

When information displayed on screen involves the entire globe, it is normally easier to use the Cartesian coordinate system, as the screen is rectangular and the x- and y-axis are well defined.

Latitude, defines from north to south of the globe, from 90° to -90°.
Longitude, defines from west to east of the globe, from -180° to 180°.
One degree is equivalent to 60 minutes, and one minutes is then equivalent to 60 seconds. One minute is also equivalent to one nautical mile, which is 1852 m at the equator.

The transformation from latitude and longitude to x- and y-value on screen is based on window coordinate in pixels. Let us define the work space with two points, start point and end point, where start point is the point on the upper left corner of the rectangle that forms the work space, end point is then the lower right corner of the rectangle. The formula is then defined as follows:

$$x = \text{width} * \left(\frac{\text{Longitude} - \text{start point's Longitude}}{\text{end point's Longitude} - \text{start point's Longitude}}\right)$$

$$y = \text{height} * \left(\frac{\text{Latitude} - \text{start point's Latitude}}{\text{end point's Latitude} - \text{start point's Latitude}}\right)$$

Transformation from x- and y-value on screen to global coordinate, can be derived from the formulas above:

$$\text{Latitude} = y * \left(\frac{\text{end point's Latitude} - \text{start point's Latitude}}{\text{height}}\right) + \text{start point's Latitude}$$

$$\text{Longitude} = x * \left(\frac{\text{end point's Longitude} - \text{start point's Longitude}}{\text{width}}\right) + \text{start point's Longitude}$$

Where x and y are the window coordinates in pixels, width and height are given in pixels and define the work space, Longitude and Latitude define a point on the globe's surface.

6.3. Zoom

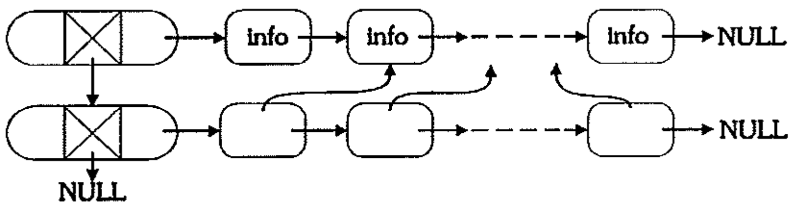


Fig. 6-2 Structure of a zoom level.

When using a graphical display, the usable amount of information presented is always limited by the size of the screen. To limit the amount of objects displayed on the screen is one of the solutions to the problem. Another solution is to let the software operator specify the area of interest and then display objects within the area only.

The latter method described is a zooming function, where an area selected will be enlarged to make full use of the entire work space to display the objects within the area. That will provide room for more detailed description for the objects.

As introduced in chapter 4, an initial network will be created during the initialization of the software. When zooming is required, the upper left corner point and the lower right corner point of the rectangle defining the zoom area will be registered and converted into global coordinate (in latitude and longitude), and stored in a head created below the pre-defined linked list. Subsequently a linear search, on the pre-defined linked lists, will take place to determine the objects within the zoom area. When an object is found, a list element will be created and a pointer will be issued from the created element to the list element that actually contains object information.

When zooming, the linear search will only be carried out on the linked list that defines the previous zoom level, since the area selected is always part of the area shown on the screen before zooming.

6.4. Symbol Design

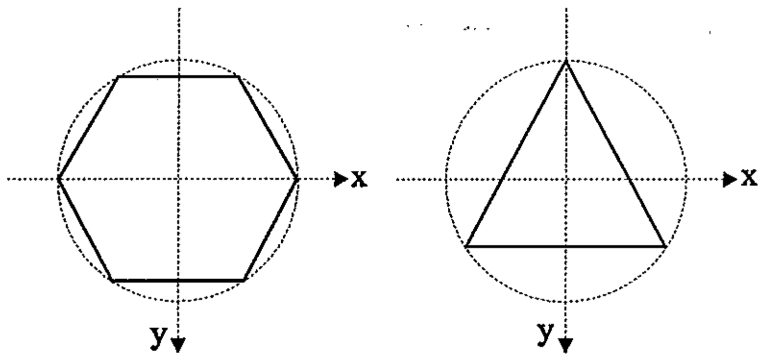


Fig. 6-3 Two types of symmetrical navaid symbols

In **Xlib** there are pre-defined functions for polygon drawings, where a record of type *XPoint* specifies the window coordinates for the corner points, defined by the software developer.

For the present project, a function that calculates the position of each corner points using trigonometric functions has implemented. The advantages of having such a routine are, firstly the size of symbol to be drawn can be made independent of the window pixel size, i.e. have a well defined size of the symbol. For instance, an airport can be set to 1 km in diameter and the size will be adjusted according to the area zoomed. Secondly, the same routine can be used for any type of symmetrical symbols.

For a symbol with odd number of corners, one of the corners will be defined on the y-axis, the trigonometric formulas are as follows:

$$x_k = r * \sin(k * \frac{2\pi}{\text{number of corners}})$$

$$y_k = -r * \cos(k * \frac{2\pi}{\text{number of corners}})$$

For symbols with even number of corners, the trigonometric formulas are as follows:

$$x_k = r * \cos(k * \frac{2\pi}{\text{number of corners}})$$

$$y_k = r * \sin(k * \frac{2\pi}{\text{number of corners}})$$

Where x and y are window coordinate, r defines the radius of the symbol to be drawn in pixels, k is an incremental variable that starts from 0 and stops at the number of corners.

6.5. Memory Reallocation

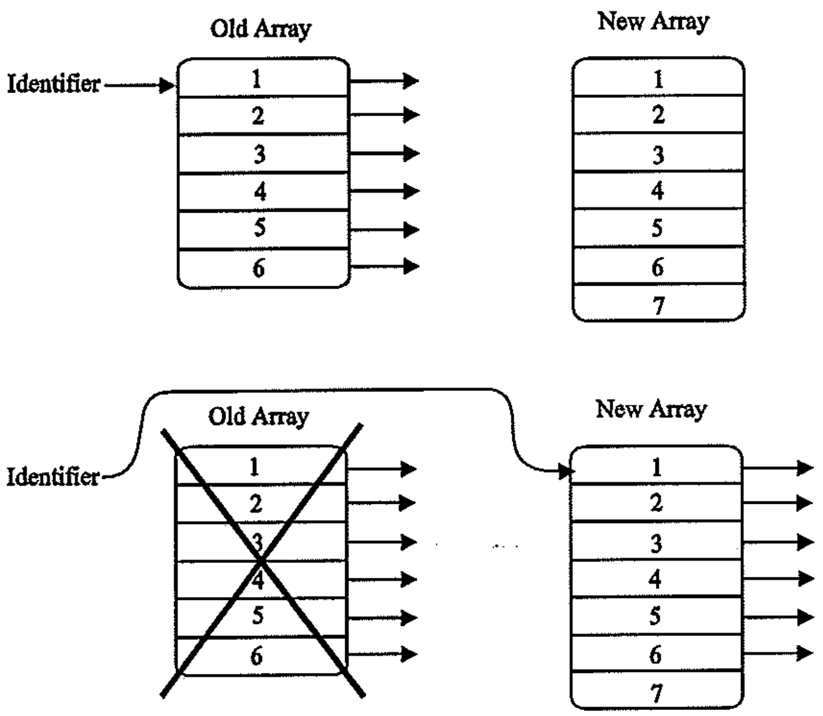


Fig. 6-4 The steps in memory reallocation.

When building up the airway network, connections will be made between the navaid objects and airway segments. When the process starts, the number of connections in each object is unknown.

An array of pointers has been chosen to keep track on each connection made in the navaid structure. Since not all objects will be assigned an airway connection, is then a waste to have a statically allocated memory space for the array in each object element. This is why the memory for each connection will be allocated individually, and then be added onto the array.

The technique used to increase numbers of elements in an array is as follows:

1. Allocate memory for a new array with a number of pointers, which is greater than the old array's.
2. Copy the contents in the old array to the new array.
3. Free the memory allocated for the old array.
4. Issue the new array the identifier for the old array.

This routine can also be used to decrease the number of elements in an array, by specify a new number which is smaller than the old one.

6.6. Summary

The above mentioned functions are just a few of many functions implemented in the system. For coordinate transformation, there is been an attempt to use spherical coordinate transformation. The advantage is when showing an optimal route, a straight line is actually being presented on the screen, since the departure node and the destination node can be placed on "equator", where the greate circle is a straight line on this map projection.

The advantage of having zoom levels is the improved speed of re-displaying the selected objects, since the pre-defined linked lists, containing objects to be displayed, are available, this will save the time spend for searching. A disadvantage is the amount of memory allocated for each zoom level, depending on the number of objects within the zoom window. This will be a serious problem when the memory resource is not large enough.

Memory reallocation is a rather time-consuming operation, using the function will effect the speed of program execution. The usage of such a routine can be avoided by allocating a memory block which is big enough to accomodate the amount of data to be stored.

7. Map Widget

7.1. Description

The Map widget is the only widget implemented for this project. It has the following functions:

- Build up the network structure.
- Display the navaid objects on the screen.
- Display object identity with the objects on the screen.
- Zoom, to show a limited area with objects.
- Retrieve information of a selected object.
- Generate an optimal route between two points.
- Generate a route manually.
- Generate a routing list in ASCII code from the route designed.
- Search for an object.

7.2. Widget Class

MapWidgetClass is the class name for Map widget, and to use an instance of the Map widget in an application, the following statement must be stated in the include statement of the application.

#include "Map.h"

7.3. Resources

7.3.1. Table Of Resources

Following resources are defined by the Map widget:

Name	Class	Type	Default
XfpNinfoCallback	XfpCInfoCallback	XmRCallback	NULL
XfpNdepartureIdAreaCallback	XfpCDepartureIdAreaCallback	XmRCallback	NULL
XfpNdestinationIdAreaCallback	XpfCDestinationIdAreaCallback	XmRCallback	NULL
XfpNrouteDistanceCallback	XfpCRouteDistanceCallback	XmRCallback	NULL
XfpNmouseMotionCallback	XfpCMouseMotionCallback	XmRCallback	NULL
XfpNworkingBoxCallback	XfpCWorkingBoxCallback	XmRCallback	NULL
XfpNwarningBoxCallback	XfpCWarningBoxCallback	XmRCallback	NULL
XfpNmessageBoxCallback	XfpCMessageBoxCallback	XmRCallback	NULL
XfpNinfoBoxCallback	XfpCInfoBoxCallback	XmRCallback	NULL
XfpNerrorBoxCallback	XfpCErrorBoxCallback	XmRCallback	NULL
XfpNforeground	XfpCForeground	XmRPixel	Black
XfpNbackground	XfpCBackground	XmRPixel	White
XfpNhighLight	XfpCHighLight	XmRPixel	Gold
XfpNmarking	XfpCMarking	XmRPixel	Red

XfpNapColour	XfpCApColour	XmRPixel	Orange
XfpNrwColour	XfpCRwColour	XmRPixel	Blue
XfpNwpColour	XfpCWpColour	XmRPixel	Green
XfpNvhfColour	XfpCVhfColour	XmRPixel	Yellow
XfpNndbColour	XfpCNdbColour	XmRPixel	Salmon
XfpNawColour	XfpCAwColour	XmRPixel	Gray
XfpNmapWidth	XfpCMapWidth	XmRInt	700
XfpNmapheight	XfpCMapHeight	XmRInt	700
XfpNsearchWindowSize	XfpCSearchWindowSize	XmRInt	200
XfpNshowObjectInfo	XfpCShowObjectInfo	XmRBoolean	False
XfpNshowAirpRway	XfpCShowAirpRway	XmRBoolean	False
XfpNshowWayPoint	XfpCShowWayPoint	XmRBoolean	False
XfpNshowVhf	XfpCShowVhf	XmRBoolean	False
XfpNshowNdb	XfpCShowNdb	XmRBoolean	False
XfpNshowAirWays	XfpCShowAirWays	XmRBoolean	False
XfpNshowAirpName	XfpCShowAirpName	XmRBoolean	False
XfpNshowWayPointName	XfpCShowWayPointName	XmRBoolean	False
XfpNshowVhfName	XfpCShowVhfName	XmRBoolean	False
XfpNshowNdbName	XfpCShowNdbName	XmRBoolean	False
XfpNshowAirWaysName	XfpCShowAirWaysName	XmRBoolean	False
XfpNsearchWindowSize	XfpCSearchWindowSize	XmRBoolean	False
XfpNsearchAirport	XfpCSearchAirport	XmRBoolean	False
XfpNserachWaypoint	XfpCSearchWaypoint	XmRBoolean	False
XfpNsearchVhf	XfpCSearchVhf	XmRBoolean	False
XfpNsearchNdb	XfpCSearchNdb	XmRBoolean	False
XfpNsearchAirway	XfpCSearchAirway	XmRBoolean	False
XfpNinputDeparture	XfpCInputDeparture	XmRBoolean	False
XfpNinputDestination	XfpCInputDestination	XmRBoolean	False
XfpNautomaticRouting	XfpCAutomaticRouting	XmRBoolean	False
XfpNautomaticZoom	XfpCAutomaticZoom	XmRBoolean	False
XfpNmanualRouting	XfpCManualRouting	XmRBoolean	False
XfpNwarningBox	XfpCWarningBox	XmRInt	0
XfpNerrorBox	XfpCErrorBox	XmRInt	0
XfpNworkingMessage	XfpCWorkingMessage	XmRString	Working
XfpNwarningOneFromOptimize RouteMessage	XfpCWarningOneFromOptimize RouteMessage	XmRString	Warning One
XfpNerrorOneFromOptimize RouteMessage	XfpCErrorOneFromOptimize RouteMessage	XmRString	Error One
XfpNerrorTwoFromOptimize RouteMessage	XfpCErrorTwoFromOptimize RouteMessage	XmRString	Error Two
XfpNerrorThreeFromOptimize RouteMessage	XfpCErrorThreeFromOptimize RouteMessage	XmRString	Error Three
XfpNerrorFrourFromOptimize RouteMessage	XfpCErrorFourFromOptimize RouteMessage	XmRString	Error Four
XfpNerrorFiveFromOptimize RouteMessage	XfpCErrorFiveFromOptimize RouteMessage	XmRString	Error Five
XfpNerrorSixFromOptimize RouteMessage	XfpCErrorSixFromOptimize RouteMessage	XmRString	Error Six

7.3.2. Resources Description

XfpNinfoCallback:

Specifies the callback to be called when a information string wished to be displayed in a Text widget.

XfpNdepartureIdAreaCallback:

Specifies the callback to be called when a departure node has been selected and its identity wished to be displayed in a TextField widget.

XfpNdestinationIdAreaCallback:

Specifies the callback to be called when a destination node has been selected and its identity wished to be displayed in a TextField widget.

XfpNrouteDistanceCallback:

Specifies the callback to be called when the accumulated route distance wished to be displayed in a TextField widget.

XfpNmouseMotionCallback:

Specifies the callback to be called when the mouse motion wished to be tracked and displayed in a TextField widget.

XfpNworkingBoxCallback:

Specifies the callback to be called when a pop up working box wished to be displayed with a message.

XfpNwarningBoxCallback:

Specifies the callback to be called when a pop up warning box wished to be displayed with a warning message.

XfpNmessageBoxCallback:

Specifies the callback to be called when a pop up message box wished to be displayed with a message.

XfpNinfoBoxCallback:

Specifies the callback to be called when a pop up information box wished to be displayed with an information message.

XfpNerrorBoxCallback:

Specifies the callback to be called when a pop up error box wished to be displayed with an error message.

XfpNforeground:

The foreground colour used in the widget workspace.

XfpNbackground:

The background colour used in the widget workspace.

XfpNhighLight:

The colour used to highlight an object selected on the screen.

XfpNmarking:

The colour used to mark out suggested airway sequence.

XfpNapColour:

The colour used to display an airport.

XfpNrwColour:

The colour used to display a runway.

XfpNwpColour:

The colour used to display a way point.

XfpNvhfColour:

The colour used to display a VHF.

XfpNndbColour:

The colour used to display a NDB.

XfpNawColour:

The colour used to display an airway segment.

XfpNmapWidth:

Specifies the width of the drawing area on screen.

XfpNmapheight:

Specifies the height of the drawing area on screen.

XfpNsearchWindowSize:

Specifies the size of the window used to search the optimal route between two nodes. Given in per cent of the smallest rectangle that encompasses the two nodes.

XfpNshowObjectInfo:

Specifies whether or not to show the information of a selected object in a text widget.

XfpNshowAirpRway:

Specifies whether or not to show airports and runways in the drawing area on the screen.

XfpNshowWayPoint:

Specifies whether or not to show way points in the drawing area on the screen.

XfpNshowVhf:

Specifies whether or not to show VHF objects in the drawing area on the screen.

XfpNshowNdb:

Specifies whether or not to show NDBs in the drawing area on the screen.

XfpNshowAirWays:

Specifies whether or not to show airway segments in the drawing area on the screen.

XfpNshowAirpName:

Specifies whether or not to display airport identifier in the drawing area on the screen.

XfpNshowWayPointName:

Specifies whether or not to display way point identifier in the drawing area on the screen.

XfpNshowVhfName:

Specifies whether or not to display VHF identifier in the drawing area on the screen.

XfpNshowNdbName:

Specifies whether or not to display NDB identifier in the drawing area on the screen.

XfpNshowAirWaysName:

Specifies whether or not to display airway identifier in the drawing area on the screen.

XfpNsearchAirport:

Specifies whether or not to search for one or several specified airports.

XfpNserachWaypoint:

Specifies whether or not to search for one or several specified way points.

XfpNsearchVhf:

Specifies whether or not to search for one or several specified VHFs.

XfpNsearchNdb:

Specifies whether or not to search for one or several specified NDBs.

XfpNsearchAirway:

Specifies whether or not to search for one or several specified airway sequences.

XfpNinputDeparture:

Specifies whether or not the selection of departure node is done.

XfpNinputDestination:

Specifies whether or not the selection of destination node is done.

XfpNautomaticRouting:

Specifies whether or not a route optimization to be taken place for the two selected nodes.

XfpNautomaticZoom:

Specifies whether or not a zoom to be taken place after optimizing the route.

XfpNmanualRouting:

Specifies whether or not a manual routing to be taken place.

XfpNwarningBox:

Specifies whether or not to display a warning box on the screen.

XfpNerrorBox:

Specifies whether or not to display an error box on the screen.

XfpNworkingMessage:

Specifies the message to be displayed when a working box appears on the screen.

XfpNwarningOneFromOptimizeRouteMessage:

Specifies the text string to be displayed when the user orders to design a route and there is a route defined previously in the memory.

The message: *Selected Route will be deleted!!!*

XfpNerrorOneFromOptimizeRouteMessage:

Specifies the text string to be displayed when the user, during route optimization, specifies two nodes that are not in any of the databases.

The message: *Can't find any of the specified nodes !!! Please RESelect BOTH departure node and destination node.*

XfpNerrorTwoFromOptimizeRouteMessage:

Specifies the text string to be displayed, when the user specifies a departure node which can not be found in any of the databases, during the route optimization.

The message: *Can't find the DEPARTURE node !!! Please RESelect the departure node.*

XfpNerrorThreeFromOptimizeRouteMessage:

Specifies the text string to be displayed, when the user specifies a destination node which can not be found in any of the databases, during the route optimization.

The message: *Can't find the DESTINATION node !!! Please RESelect the destination node.*

XfpNerrorFourFromOptimizeRouteMessage:

Specifies the text string to be displayed, when the user specifies two nodes which do not have any airway connections, during the route optimization.

The message: *Nodes selected have no airway connections !!! Please RESelect another pair of navaids.*

XfpNerrorFiveFromOptimizeRouteMessage:

Specifies the text string to be displayed, when the user specifies a departure node which does not have any airway connection, during route optimization,.

The message: *The departure node selected has no airway connections !!! Please select another navaid that has at least one airway connection.*

XfpNerrorSixFromOptimizeRouteMessage:

Specifies the text string to be displayed, when the user specifies a destination node which has no airway connection, during route optimization.

The message: *The destination node selected has no airway connections !!! Please select another navaid that has at least one airway connection.*

7.4. Functionality

7.4.1. Widget Methods

The widget methods are needed for making a functioning widget. **Initialize Method**, **Expose Method**, **Set_Values Method**, and **Destroy Method** are the methods implemented in the Map Widget.

```
static void Initialize (request, new, args, num_args)
Widget      request, new;
ArgList     args;
Cardinal    *num_args;
```

This function belongs to the **Initialize** method, and is called when the application calls *XtVaCreateManagedWidget*. It contains codes for setting the initial values of private instance variables, e.g. creating the airway structure and network, checking to make sure that the values of the public instance variables are valid.

```
static void Redisplay (request, new, args, num_args)
Widget      request, new;
ArgList     args;
Cardinal    *num_args;
```

This function belongs to the **Expose** method, and is responsible for initially drawing into a widget's window and for redrawing the window every time a part of the window becomes exposed. This redrawing is necessary because the X server does not maintain the contents of windows when they are obscured. When a window becomes visible again, it must be re-drawn.

```
static Boolean SetValues (request, new, args, num_args)
Widget      request, new;
ArgList     args;
Cardinal    *num_args;
```

This function belongs to the **Set_Values** method, and is called when the application calls *XtVaSetValues* to change widget resources during run time. It will validate the values of the public instance variables, and recalculates the private variables that depend on the public variables that have changed.

```
static void Destroy (request, new, args, num_args)
Widget      request, new;
ArgList     args;
Cardinal    *num_args;
```

This function belongs to the **Destroy** method, and is called when the application calls *XtDestroyWidget*, to free the memory allocated by the Map Widget.

7.4.2. Widget Actions

Widget actions are procedures responding events caused by an standard input device, a keyboard or a mouse. An action translation table is defined in the Map Widget to issue the function that responds when a certain action takes place.

The following Widget action has been implemented:

```
static void MouseCoord (w, event, params, num_params)
Widget       w;
XButtonEvent *event;
String       *params;
Cardinal     *num_params;
```

This function will respond to the movement of the mouse, i.e. when the mouse is within the working area, an event will be generated as soon the mouse moves. The function will then call for a *callback function* to display the mouse position in a TextField Widget.

```
static void StartAction (w, event, params, num_params)
Widget       w;
XButtonEvent *event;
String       *params;
Cardinal     *num_params;
```

This function will respond when the left mouse button has been pressed down. It registers the position of the mouse.

```
static void MoveAction (w, event, params, num_params)
Widget       w;
XButtonEvent *event;
String       *params;
Cardinal     *num_params;
```

This function will respond to the movement of the mouse while the left mouse button is being pressed. It displays a rectangle with the left upper corner stationed at the position registered in *StartAction*, and right lower corner defines by the current mouse position. The rectangle displayed will have the same height-width relation as the workspace.

```
static void CompleteAction (w, event, params, num_params)
Widget       w;
XButtonEvent *event;
String       *params;
Cardinal     *num_params;
```

This function will respond when the left mouse button is being released. The position of the mouse cursor will then be registered and together with the point registered in *StartAction* form a rectangle that defines the area to be zoomed. If the rectangle does not exceed an 4X4 pixels square, the zooming function will not be carried out. If the mouse position has not been changed since *StartAction*, the widget will then search for an object that has its window coordinate within a 4X4 pixels square, where the mouse cursor position defines the centre of the square, highlight it and display its identity when found.

```
static void RegretAction (w, event, params, num_params)
Widget        w;
XButtonEvent *event;
String       *params;
Cardinal     *num_params;
```

This function will respond when the middle mouse button is being pressed and released. It will perform the zoom up action, and releases the allocated memory for the linked lists that were created during the zoom in action.

```
static void ConfirmInput (w, event, params, num_params)
Widget        w;
XButtonEvent *event;
String       *params;
Cardinal     *num_params;
```

This function will respond when the right mouse button is being pressed and released. It confirms an object that has been selected, to be stored in structure for route designing purpose.

7.5. Exported Functions

```
Map_OptimizeRoute(w, departure, destination)
Widget    w;
char      *departure, *destination;
```

Activates the route optimizing routine in Map Widget. Departure and destination specifies the identity of the two selected nodes.

```
Map_AutomaticRouting(w)
Widget    w;
```

Confirms the deleting of a pre-defined route and the route optimizing routine will be carried out.

```
Map_ManualRouting(w)
Widget    w;
```

Activates the manual routing option.

```
Map_ManualRoutingCompleted(w)
Widget    w;
```

Notify that the manual routing has completed.

```
Map_DrawSelectedRoute(w)
Widget    w;
```

Displays the selected route on the screen.

Map_SaveRouteList(w)
Widget w;

Save a pre-defined route as an ASCII file.

Map_RetrieveRouteList(w)
Widget w;

Interpret a saved route, an ASCII file, in such a way that the program can reprocess the information, in order to display the route on the screen.

Map_FreeHash(w)
Widget w;

Activates a routine in Map widget, to release the allocated hash tables.

7.6. File Output

The ultimate task of this project is to generate an optimal flight planning route, that can be printed out for the clients disposal.

Once a route has been designed, either automatically or manually, there is a linked list created with elements pointing to the navaid posts. This forms the specified routing list. The routing list can then be saved as ASCII text file, under specified company, called company route.

An example on a generated routing list from Malmö Sturup to Stockholm Arlanda:

OPTIMAL FLIGHT ROUTE FROM ESMS TO ESSA
PRODUCED BY FLYGPRESTANDA AB.
ALL RIGHT RESERVED.

Navaid Identity	Airway Identity	Distance Remained	Distance To Next Navaid	Course
ESMS	-	225 NM	10 NM	-
OE	R59	215 NM	45 NM	31°
ASNEN	R59	170 NM	40 NM	31°
REN	R59	130 NM	22 NM	29°
REN22	R59	108 NM	57 NM	31°
GUNNI	R59	51 NM	41 NM	8°
BOVEN	BOVE1E	10 NM	10 NM	-
ESSA	-	0 NM	0 NM	-

7.7. Summary

Map Widget is the only widget implemented for this project, it handles all functions needed for the flight planning system and provides communication between the databases and the user interface.

No doubt the Map Widget is quite complex, but it still lacks of some necessary functions to make it fully operational. For instance, a function to update the databases is highly required for a system under operation, a routing list interpreter to show a pre-defined route on the screen, world vector shoreline to provide users the accurate geographical orientation. The list can be made long, but due to the limitation of time and the size of the project, the functions mentioned above will not be included in this project.

References

Adrian Nye, 1991. "Xlib Programming Manual". O'Reilly & Associates, Inc.
ISBN: 0-937175-11-0

Adrian Nye and Tim O'Reilly, 1992. "X Toolkit Intrinsics, Programming Manual". O'Reilly & Associates, Inc.
ISBN: 0-937175-62-5

Open Soft Foundation, 1991. "OSF/Motif Programmer's Guide". Prentice Hall.
ISBN: 0-13-640673-4

8. Application

8.1. Description

The application program contains codes for realizing widgets, and handles communication between them.

For Widget Hierarchy please refer to Appendix A.

8.2. Widgets

8.2.1. Main Menu

The main menu appears together with the graphic work space when the software has been initiated.

The main menu consists of push- and toggleButtons to "pop up" and "pop down" submenus. It also contains two textFields for displaying the mouse position in latitude and longitude. The functions of the menu buttons are described as followed:

- A pushButton for exiting the application.
- A toggleButton to enable and disable the appearance of the *AIRPORT* navaid objects on screen.
- A toggleButton to enable and disable the appearance of the *WAY POINT* navaid objects on screen.
- A toggleButton to enable and disable the appearance of the *VHF* navaid objects on screen.
- A toggleButton to enable and disable the appearance of the *NDB* navaid objects on screen.
- A toggleButton to enable and disable the appearance of the *AIRWAY* connections on screen.
- A toggleButton to enable and disable the appearance of *all* navaid objects and *AIRWAY* connections on screen.
- A toggleButton to pop up and pop down the *route Design* submenu.
- A toggleButton to pop up and pop down the *show Info.* text box.
- A toggleButton to pop up and pop down the *search Object* submenu.
- A toggleButton to pop up and pop down the *show ID* submenu.
- A textField for displaying the mouse position in *latitude.*
- A textField for displaying the mouse position in *longitude.*

8.2.2. The Route Design Submenu

The route optimize submenu will appear as soon the "*Design A Route*" toggleButton on the main menu has been selected, and disappears when the toggleButton has been released.

The submenu consists of a set of pushButtons and a textField for displaying the route distance. The pushButtons' functions are as followed:

- A pushButton to activate a submenu for *automatic route optimization.*

- A pushButton to activate the manual route design function.
- A textField for displaying the distance of the route designed in nautical mile.
- A pushButton to confirm a navaid selected to be included into the manual designed route structure.
- A pushButton to activate a *fileSelectionBox* for retrieving a pre-defined route in ASCII form.
- A pushButton to activate a *fileSelectionBox* for saving the designed route in ASCII format.
- A pushButton to activate the function of showing a pre-defined route on screen.

8.2.2.1. Automatic Route Optimization

The Automatic Route Optimization submenu will appear on the screen, as soon the pushButton for automatic routing, in the route design submenu, has been pressed. It contains:

- A textField for displaying the identity of the departure navaid for route optimization.
- A textField for displaying the identity of the destination navaid for route optimization.
- A pushButton to confirm the departure navaid selected.
- A pushButton for displaying the route designed with zoom function.
- A pushButton for displaying the route designed without zoom function.

8.2.3. Info Window

The Info Window will appear on the screen when the toggleButton "*Show Info*", in the main menu, is being selected, and will disappear when the toggleButton is being released. The Info Window is for displaying all the information stated in databases about a specified navaid object.

8.2.4. The Search Object Submenu

The Search Object Submenu will appear on the screen when the toggleButton "*Object Search*", in the main menu, is being selected, and will disappear when the toggleButton is being released. The submenu contains:

- A set of toggleButtons for selecting the type of navaid to be sought for, one toggleButton for each type of navaid.
- A textField for input and display the navaid identity for the navaid to be sought for.
- A textField for input and display the navaid area code for the navaid to be sought for.
- Two textField for input and display the position of the navaid to be sought for.

8.2.5. The Show ID Submenu

The Show ID submenu will appear when the toggleButton "*Show ID*", in the main menu, is being selected, and will disappear when the toggleButton is being released. The submenu contains a set of toggleButtons for selecting the type of navaid of which the identity wishes to be displayed on the screen together with the navaid symbol. There is one toggleButton for each navaid.

References

Adrian Nye, 1991. "Xlib Programming Manual". O'Reilly & Associates, Inc.
ISBN: 0-937175-11-0

Adrian Nye and Tim O'Reilly, 1992. "X Toolkit Intrinsics, Programming Manual".
O'Reilly & Associates, Inc.
ISBN: 0-937175-62-5

Open Soft Foundation, 1991. "OSF/Motif Programmer's Guide". Prentice Hall.
ISBN: 0-13-640673-4

9. Discussion & Experience Gained

9.1. Description

For project developing, it is inevitable to come across situations where a wrong or less appropriate solution has been introduced to a certain problem, and sometimes there may be many alternative solutions to the problem.

9.2. Database Interpreter

For each type of navaid object, there is one database interpreter implemented. This is due to the information contained in each of the database are rather different, the length of post element varies for each type of navaid object.

9.3. Algorithm for Route Optimization

9.3.1 Experience Gained

A recursive method has been chosen for the implementation of the route optimizing routine. As a first attempt to design an optimum route, the algorithm described in chapter 5, was used. To limit the number of visits at each navaid point and to avoid locally formed loops, airway segments were closed as soon as the recursion passes through them. The consequence of this action results in uncertainty of distance accumulated. When recursion is carried out, all airway segments will be visited twice exactly, since the airway goes in both directions. Due to the network structure, a navaid object may then be visited more than once in the same branch of recursion. In this case the accurate distance would not be determined until the recursion of the airway segment has been fully terminated.

The following conditions have to take into consideration when implementing the algorithm:

1. The way back can never be the shortest.
2. An airway segment can only be closed when the distance determined are selected from airway segments that are already closed, except for the way back.
3. An error in a determined distance will always propagate further in a recursion.

As a simple criterion to decide whether or not the iteration should carry on, is to determined the number of distances altered in the airway network.

9.3.2 Dynamic Programming

As mentioned earlier, the algorithm used for implementing the route optimize routine, is a variation of *Dynamic Programming*. Dynamic Programming uses the principal of solving subproblems recursively derived from the original problem. The subproblems are not independent of each other, i.e. the subproblems share subproblems. To limit the number of times solving the same subproblem, the dynamic programming algorithm saves the result of each subproblem solved, in a table. This actually involves an enormous amount of storage for the results from all the subproblems, although the problem may be moderately complicated.

As for this project, the majority of the memory on board has been used up when airway network is initiated, the amount of memory remain is rather limited. So, an application of the conventional dynamic programming algorithm on route optimization is not practical unless an investment has been made to increase the memory on board. Otherwise it will result in the computer using its swap area and cause the time taken to optimize a route to be unacceptable.

References

Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, 1991, "Introduction to Algorithms". The MIT press and McGraw-Hill Book Company.
ISBN: 0-262-03141-8 (MIT press). ISBN: 0-07-013143-0 (McGraw-Hill).

R. Bellman & S. Dreyfus. NJ, 1962, "Applied Dynamic Programming".
Princeton University Press, Princeton.

10. Conclusion

The Flight Operation Support System has implemented the way Flygprestanda AB specified. The function for showing runways and the flight procedures used for takeoff and landing at the airport is not implemented due to the time taken for the other functions, also due to lack of the memory resources in the computer where this software package is developed at the moment.

There is no help function available in the system, due to all concentration was emphasized on the functionalities of the application..

The choice of using **OSF/MOTIF™** widget set to provide an graphical user interface turned out to be excellent. The effort required to expand the user interface with a pushButton or pop up menu is rather minor. The only drawback would be the portability of the software is limited to computers having **OSF/MOTIF™** installed.

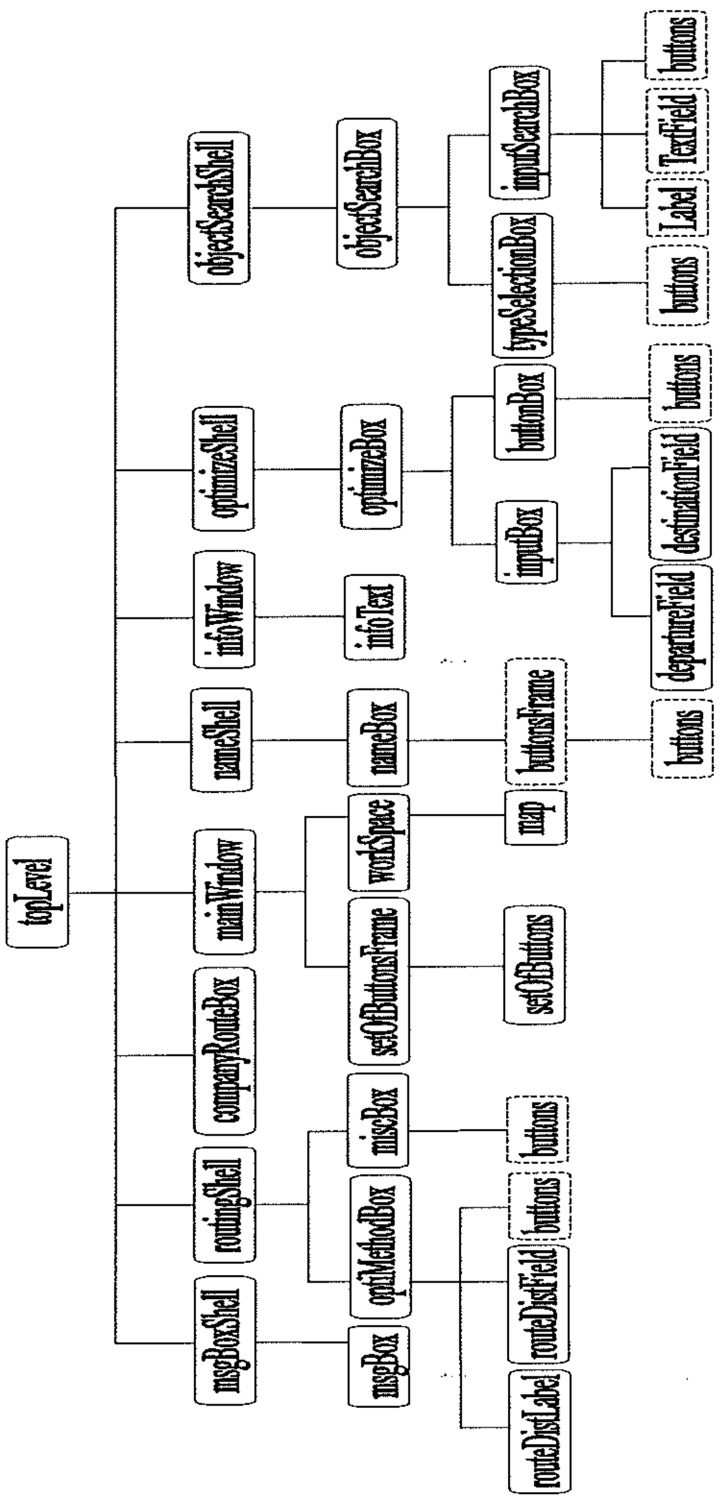
No doubt the software package has completed according to the specification listed in chapter 1, but it still lack of several functions in order to be used for comercial purposes. Functions like:

- aircraft specific data to calculate the fuel and time consumed for a specific route.
- preferential routes (standard routes) for countries that have such.
- different map projections, e.g. Albers equidistant map projection.
- Extended Range Operations, permission to fly a certain flight-time away from the nearest airport with a two engine aircraft.
- a function to update the airway network while the software is active.
- a function to update the databases.
- wind and weather.

are to be implemented in the future. The result of this project will provide a foundation for further developement of the system.

Appendix A

Widget Hierarchy



Flight Operations Support System User's Guide

Table Of Contents

1. Introduction

This manual is an attempt to provide adequate information for the operation of the **Flight Operations Support System** software package, **Navigate**. It includes instructions concerning route optimization, object selecting, display object information etc.

The program is implemented in such way that the windows displayed on screen can be placed anywhere on the screen and moved freely.

The software package is implemented under the programming language **C**, using **OSF/MOTIF**™ to provide a graphical user interface.

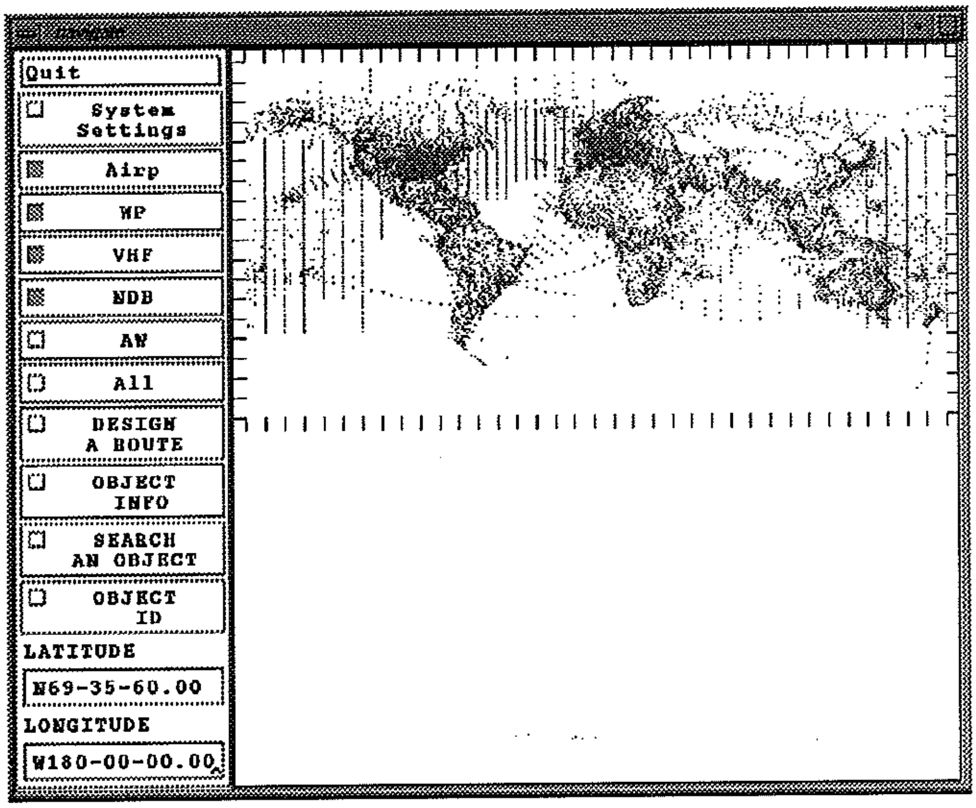
The hardware requirements are:

- A computer with a computing power comparable to at least an **Intel**™ 80386 processor.
- At least 20 Mb of internal memory.
- A graphic display.
- A pointing device is not a must, but is highly recommended.

The software requirements are:

- **UNIX**™ operating system installed and running.
- **OSF/MOTIF**™ software package installed.

2. Initial State



LATITUDE
N69-35-60.00
LONGITUDE
W180-00-00.00

Fig. 2-1 The appearance of the program window.

When starting up the software, the application will initialize the entire airway network structure and place it in the primary memory area to shorten the time taken for different operations. This will take about 1½ to 2 minutes from the moment you start the software till the program window appears on screen.

The window presented on the screen, has a graphic display area, called workspace, and a command button menu. The command button set consists of 6 toggleButtons with direct application command connected, and 4 toggleButtons with connected submenus. When user selects one of the command buttons that has a submenu connected, a submenu will pop-up on the screen. Submenus will appear as a stand-alone set of buttons and can be moved and placed freely on the screen.

2.1. Mouse Button Functions

The following functions are available throgh the mouse buttons, when a three-button mouse is part of the hardware equipment.

- Left mouse button: functions like object selection, zoom and command button selection are tied to this mouse buttton.
- Middle mouse button: cancels the zooming function.
- Right mouse button: confirms an input.

More detailed descriptions are given together with the respective functions.

3. The Command Button Menus

The command button set will always appear to the left of the working area. It includes the following functions:

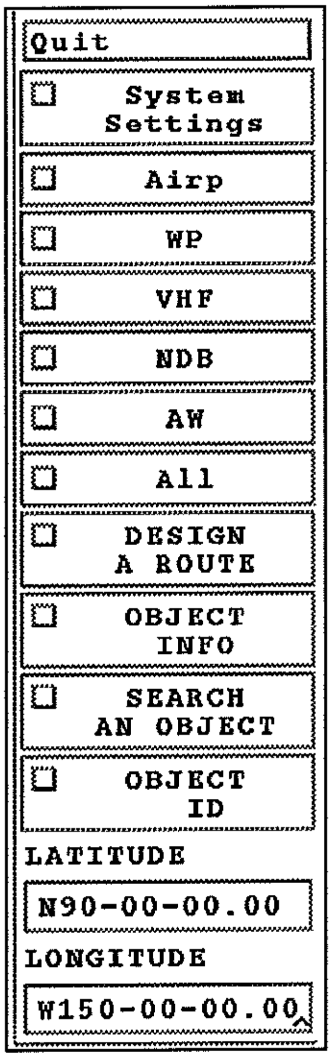


Fig. 3-1 The Command Button set

- Quit, to end the program session.
- System Settings, a submenu appears when this toggleButton has been pressed, where some system settings can be changed, e.g. the objects' colour. The submenu can be placed anywhere on the screen.
- Airp, show airports as symbols on the working area.
- WP, show way points as symbols on the working area.
- VHF, show both VOR and DME as symbols on the working area.
- NDB, show NDBs as symbols on the working area.
- AW, show Airways as line segments on the working area.
- All, show all the navaid objects as symbols on the working area.
- DESIGN A ROUTE, toggleButton that activate a submenu, where automatic and manual routings methods can be chosen. The submenu can be placed anywhere on the screen.
- OBJECT INFO, toggleButton that activates a text window for displaying the object information. The text window can be placed anywhere on the screen.
- SEARCH AN OBJECT, toggleButton that activates a submenu for searching an object, where different search criteria can be chosen. The submenu can be placed anywhere on the screen.
- OBJECT ID, toggleButton that activates a submenu, where object id for a particular type of navaid object can be chosen to be displayed on the working area. The submenu can be placed anywhere on the screen.

3.1. The Coordinate Windows

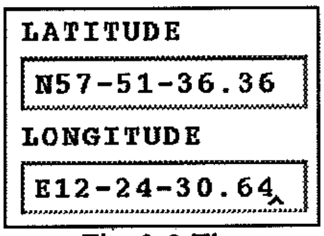

LATITUDE
N57-51-36.36
LONGITUDE
E12-24-30.64

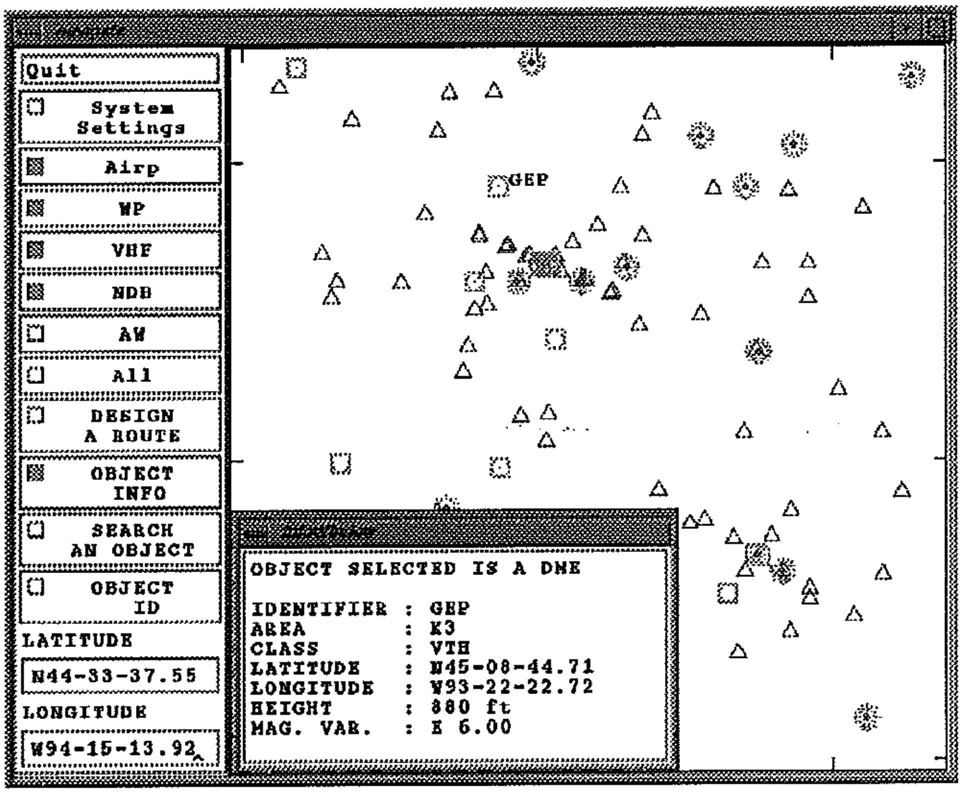
Fig. 3-2 The Coordinate Window.

The coordinate window shows the current position of the mouse cursor in latitude and longitude. The figures will be changed as soon as the pointing device has changed its position within the workspace.

The range of figures displayed are, S90-00-00.00 to N90-00-00.00, for latitude and W180-00-00.00 to E180-00-00.00, for longitude.

i. The initials S, N, W and E stand for south, north, west and east.
ii. The first two digits refer to degrees of either latitude or longitude.
iii. The next two of digits refer to minutes of either latitude or longitude.
iv. The last four digits refer to seconds with two decimals of latitude or longitude. 0.01 second of longitude corresponds to 0.3087m.

4. Zoom & Select



LATITUDE
N44-33-37.55
LONGITUDE
W94-15-13.92

```
OBJECT SELECTED IS A DME

IDENTIFIER  : GEP
AREA        : K3
CLASS       : VTH
LATITUDE    : N45-08-44.71
LONGITUDE   : W93-22-22.72
HEIGHT      : 880 ft
MAG. VAR.   : E 6.00
```

Fig. 4-1 Navaids are presented as cartographic symbols.

The zooming function is always active but can only be used when a pointing device, with three buttons, is included as part of the hardware equipment. When using a three-button mouse, simply press down the left mouse button to activate the zoom function, this will define the upper left corner of the zoom window. Without releasing the mouse button, drag the mouse either towards the right, downward or from left to right diagonally downwards, a rectangle made up of dashed lines will appear on the working area. As the mouse cursor is dragged further, the rectangle will increase its size according to the mouse movement. When the rectangle has enclosed the area to be zoomed, release the left mouse button. The workspace will first be cleared and then display the objects within the area zoomed.

The dash-lined rectangle that forms the zoom window, has an absolute width and height relation to ensure the proportion of the area zoomed. To zoom out, press down and release (click) the middle mouse button on a three-button mouse.

The position of the upper left corner of the zoom window can be changed, if the area to be zoomed is not enclosed in the zoom window. By moving the mouse cursor back to the point that defines the upper left corner of the zoom window, without releasing the left mouse button, move the mouse cursor beyond the point, moving either toward the left, up or from right to left diagonally upward. Stop at the position where the area to be zoomed is to the right and below of the mouse cursor, then without releasing the left mouse button, move the pointing device toward the right or downward to form the zoom window desired.

To cancel the zooming action, move the mouse cursor back to the starting point and release the left mouse button.

When the program has been initiated, all navaid objects are displayed as coloured dots, this is due to the large number of navaid objects available, otherwise the screen will appear cluttered up. When zoom has been utilized, the number of objects will decrease for every zoom level, that has been formed. When the number of objects available has decreased to a certain extend, the objects will then be presented as coloured squares. If zoomed in further, (when a pixel value is equivalent to 1 km) the navaid objects will then be presented as cartographic symbols.

When both objects are presented as coloured squares and as cartographic symbols, the objects are then sensitive for selection, i.e. information about an object displayed on the working area can be retrieved by using the pointing device to point and select. The object selection is done by moving the mouse cursor to where the object to be selected is situated, press down and release (click) the left mouse button. The object selected will now be highlighted, being displayed with another colour, and the object identity will be displayed to the right of the selected object.

As a simple guide for the functions described above, a step by step description is as follows:

Zoom:

1. Define the upper left corner of the desired zoom area at the mouse cursor by pressing down the left mouse button.
2. Drag the mouse towards the right, without releasing the mouse button.
3. When the dash-lined rectangle has enclosed the area to be zoomed, release the left mouse button.

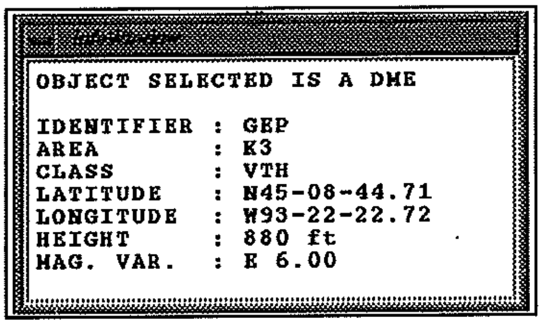
Cancel Zooming:

- Press and release (click) the middle mouse button.

Select an object:

1. Zoom in the area where the object or objects to be selected is located. The selecting function is not activated until the objects on screen are presented as either cartographic symbols or coloured squares. Zoom in another time if necessary.
2. Move the mouse cursor to where the object symbol is situated and click with the left mouse button.

4.1. Object Information



```
OBJECT SELECTED IS A DME

IDENTIFIER  :  GEP
AREA        :  K3
CLASS       :  VTH
LATITUDE    :  N45-08-44.71
LONGITUDE   :  W93-22-22.72
HEIGHT      :  880 ft
MAG. VAR.   :  E 6.00
```

Fig. 4-2 Info Window.

If more detailed information about an object to be retrieved is desired, the object info window will have to be initiated. By using the pointing device to place the cursor on the toggleButton labeled "OBJECT INFO" in the command button set, the info window will appear on screen. Follow the instructions described above to select an object from which the information to be displayed is desired, on the info window.

4.2. Object Identity

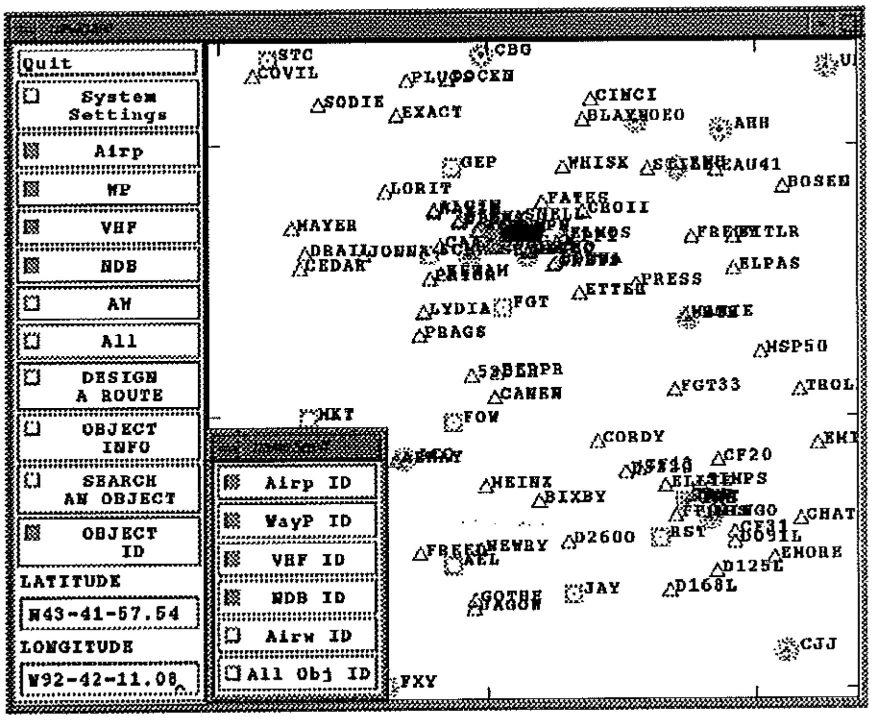


Fig. 4-3 The main window with Object Id submenu.

To facilitate the selecting of navaid object, a function showing objects' identities can be activated. The identities will only appear on those objects that are already showing in the workspace. To activate the function, select the toggleButton with label "OBJECT ID", and a pop-up menu will then appear on the screen. To display the object identity for an object type, simply select the toggleButton with the object type labeled. The object identities will appear to the right of each object symbol, when the action is carried out. To take away the object identities, press the same toggleButton again.

5. Designing A Route

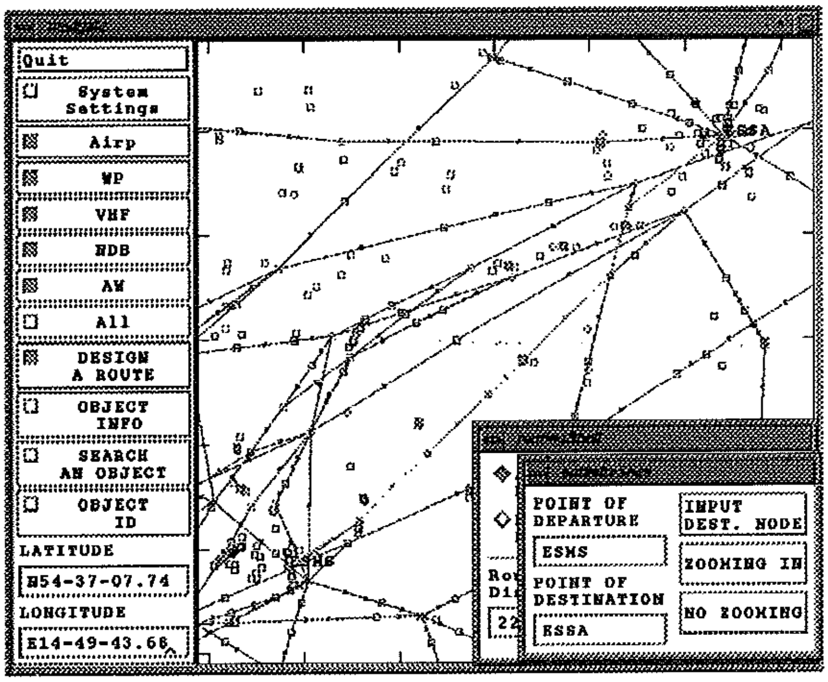


Fig. 5-1 A optimized route between Malmö Sturup and Stockholm Arlanda in Sweden.

The route optimization is divided into two parts. An automatic route optimization, and a function for manual route design. Both functions will be available through selecting the toggleButton labeled "DESIGN A ROUTE".

5.1. Automatic Route Optimization

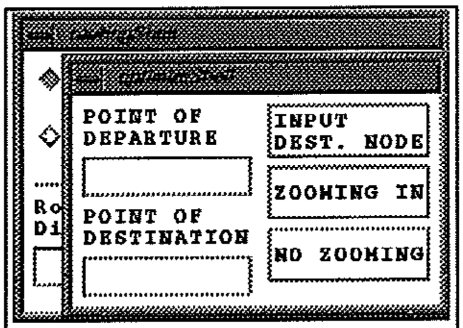


Fig. 5-2 Pop up menu for route optimization

To design an optimal route through the route optimize function, follow the steps described below:

1. Select the toggleButton labeled "DESIGN A ROUTE" on the command button set.

2. A menu will appear on the upper left corner of the screen (see fig. 5-4), select the button labeled "Automatic Routing" to pop up the final submenu for route optimization.

3. Specify two point of which the route optimization is to be performed. There are two ways of specifying the departure and destination point.

- Select one of the textFields by clicking in the textField area with the left mouse button. Type in the identity of the departing object and the destination object in the textField respectively.

- Select the objects by using the object selection function described in chap. 4. For the departure node, the identity of the object selected will appear in the " Point of departure" textField. The object selected as the departing point can be changed as long as the input has not been confirmed. Confirmation of the departure node is done by either clicking with the right mouse button or pressing the pushButton labeled "INPUT DEST. NODE" in the pop up menu shown in fig 5-3. For the destination node, the selection is identical to the departure node, with the destination object identity displayed in the "Point of destination" text filed.

The two input methods can be combined, i.e. selecting one of the nodes by using the pointing device and type in the navaid identity for the other.

4. Start the route optimization by either pressing the pushButton labeled "ZOOMING IN", to display on the area involved for the route optimization, or pressing the pushButton labeled "NO ZOOMING" to preserve the area shown before route optimization.

5. The time taken for route optimization varies depending on the complexity of the airway network involved. When the entire airway network is used for route optimization, it takes about 3 minutes for a computer with an i80486 processor. When the optimization is completed, the optimal route will be displayed on screen.

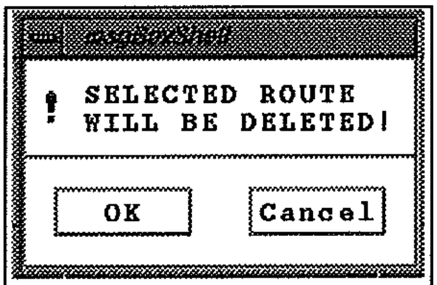

SELECTED ROUTE WILL BE DELETED!

Fig. 5-3 A warning box

If the nodes specified have no airway connections, a message box will appear in the middle of the screen.

A route designed will remain in the primary memory until the next time the optimizing function is called. A warning box will appear when it occurs, the user is then given the choice to either continue with the route design or to cancel the operation to have the chance to save the route designed.

The route designed can be displayed by selecting the pushButton labeled "SHOW SELECTED ROUTE" on the routing box.

The distance from the departure node to the destination node in nautical mile is displayed in the "Route distance" textField.

5.2. Manual Route Design

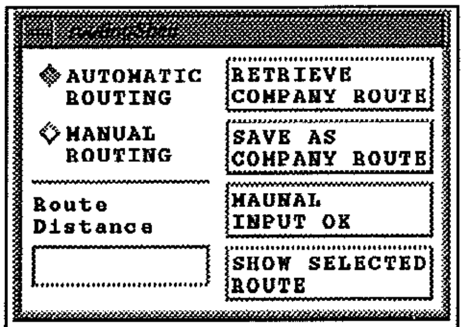


Fig. 5-4 Routing Pop up Menu

To design a route manually follow the steps described below:

1. Select the toggleButton labeled "DESIGN A ROUTE" on the command button menu. A pop up menu appears on the screen.

2. Select the toggleButton labeled "MANUAL ROUTING" on the Routing Box that has appeared.

3. Specify the point of departure by using the mouse and select the desired object's symbol on screen. The selection is not confirmed until the user clicks the right mouse button, i.e. the user can regret the selection by selecting another object.

4. Display the airways on the workspace by selecting the toggleButton labeled "AW", if the airways are not displayed.

5. Select an airway by clicking on the square in the middle of airway segments that is closest to the object selected. The airway will then be highlighted from the segment that is connected to the last connecting node of the airway sequence.

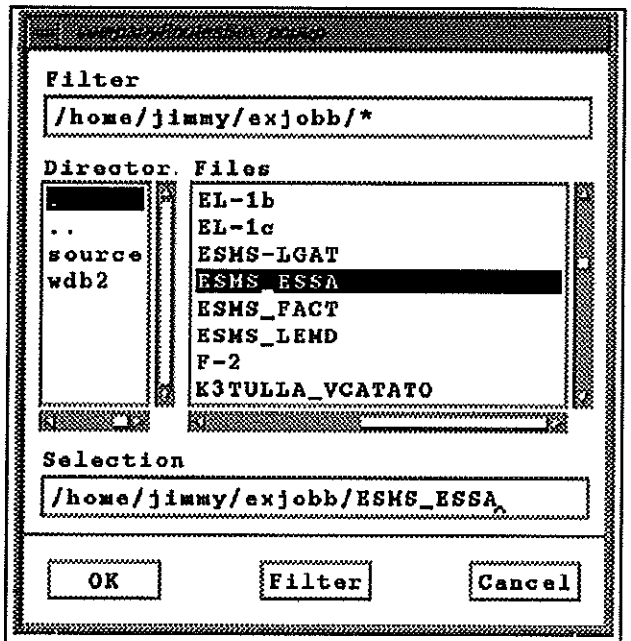
6. To confirm the airway selected, the user has to select an object that is connected to the selected airway. This is normally the intersection of the airway selected and the airway to be selected for the next segment.

7. Repeat the steps 5 and 6 until the destination node is reached.

The route designed can be displayed by selecting the pushButton labeled "SHOW SELECTED ROUTE" on the routing box.

The distance from the departure node to the destination node in nautical mile is displayed in the "Route distance" textField.

5.3. Save and Retrieve a Route



```
Filter
/home/jimmy/exjobb/*

Director. Files
          EL-1b
..        EL-1c
source    ESMS-LGAT
wdb2      ESMS_ESSA
          ESMS_FACT
          ESMS_LEMD
          F-2
          K3TULLA_VCATATO

Selection
/home/jimmy/exjobb/ESMS_ESSA

OK        Filter        Cancel
```

Fig. 5-5 A FileSelectionBox

A route designed can be saved as an ASCII text file. This is accomplished by selecting the pushButton labeled "SAVE AS COMPANY ROUTE" in the routing box. A fileSelectionBox will then appear on screen, where the user can specify the file name of the text file created.

A filename can either be selected from the file list in the fileSelectionBox, if the filename is pre-defined, or type the filename in the Selection textField if the filename is nonexistent.

To retrieve a pre-defined route, simply press the pushButton labeled "RETRIEVE COMPANY ROUTE" in the routing box. A fileSelectionBox will then appear on screen, where the route to be retrieved can be selected either through press and select from the file list or by typing the filename in the Selection textField.