Tallinn Technical University

Functions for Handling Digital Maps

Diploma Project
by

Hanno Kuus

Tallinn 1998

Of meridians, and parallels, man hath weawed out a net, and this net thrown upon the heavens, and now they are his own.

–John Donne, 1611

Abstract

Functions for Handling Digital Maps

Hanno Kuus

Tallinn Technical University 1998

In today's computerized world most of old-fashioned cartographic data is rendered obsolete, being revolutionized by the era of digital mapping. As with nearly every new invention that considerably enlarges current potentials, the lack of modern tools had been somewhat slowing down the use of digital maps in the past. This issue is addressed by many different instruments that keep appearing now and then, and today there are many excellent tools available for processing spatial information.

Current work gives overview of the Mapper function library, being developed for easy integration of chart information into miscellaneous computer applications. It provides several subroutines that can be used as a base for specific Geographic Information Systems (GIS), built to handle nautical charts in digital form.

At the beginning reader is got acquainted with basic concepts of cartography and geodesy, several map projections are also presented. Also in this chapter specialties of nautical charts and relevant international standards are discussed.

The main part describes the development process of Mapper, from design to realization. Several aspects, with both problems and solutions, are considered, together with more practical discussion about the usage of provided functions. Examples of working systems that use Mapper as a map engine are also provided.

Kokkuvõte

Digitaalkaartide kuvamise ja redigeerimise funktsioonid

Hanno Kuus

Tallinna Tehnikaülikool 1998

Tänapäeva moodsas maailmas leiavad arvutid üha laiemat rakendust, üheks suureks valdkonnaks, kus arvutite abil on toimunud revolutsioon, on ka kartograafia. Praegusel ajal on praktiliselt kogu kasutatav ruumiline informatsioon koondatud numbrilisel kujul raalidesse, muutes kaardiinfo töötlemiseks mõeldud tavapärased vahendid kasutuskõlbmatuks. Geograafiliste andmete arvutites kasutamiseks on maailmas loodud palju uusi abivahendeid, millest ühte tutvustatakse antud kirjutises.

Käesolev töö annab ülevaate kartograafilise informatsiooni esitamiseks ning modifitseerimiseks loodud rakendusfunktsioonide teegist Mapper. Tegemist on alamprogrammide kogumiga, mis võimaldab küllaltki lihtsate vahenditega luua spetsiifilisi, digitaalsetel merekaartidel põhinevaid geograafilisi infosüsteeme (GIS).

Esimeses osas on toodud ülevaade geodeesia ning kartograafia põhimõistetest ning tutvustatud mõningaid kartograafilisi projektsioone. Lisatud on ka põgus ülevaade merekaartidest ning valdkonda reguleerivatest rahvusvahelistest standarditest.

Töö põhiosa keskendub teegi Mapper spetsifitseerimise ja realiseerimise aspektidele, andes ülevaate nii tekkinud probleemidest kui ka neile leitud lahendustest. Tuuakse ka juhiseid loodud protseduuride kasutamiseks. Lõpetuseks on toodud näiteid loodud funktsioonide kasutamisest erinevates rakendusprogrammides.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In today's fast-moving environment computers are used in about every field of human activity. Modern computers can do a lot more than their predecessors, which were capable of doing only simple calculations, yet there are tasks that digital machines cannot perform even nowadays. One of the fields that has taken off quite recently is the usage and visualization of spatial data — the *Geographic Information Systems* (GIS). The first GISes started up at the sixties and spread more widely in 1970s and 1980s as hardware got more powerful (and cheaper) and now everyone can use such systems even on personal computers. Geographically referenced data is used nowadays not only by cartographers but also by very wide range of specialists. The sector of Geographic Information Systems is still one of fastest growing in computer industry with growth of about 20 per cent each year [9]. This is a clear indication that the case for GIS is not "closed", active studies for both theoretical and practical aspects are done every year, helping to push today's technology into tomorrow.

This paper is written to describe the production of a toolbox, called the Mapper library. The development of this library started in 1995 and is presently held by the company *R-Süsteemid*. The set contains handful of subroutines, written in *C* programming language for *X Window System*, providing the capability of viewing and editing nautical charts. These functions, originally developed for *Estonian National Maritime Board* (ENMB), give to their user powerful tool for creating GISes of various kinds.

Author of this work, as a software engineer at the R-Süsteemid company, has been involved with development of Mapper from the beginning, participating in design stage as well as writing fair amount of source code for the software. Presentation model and aspects of its realization have been the main area of interest for me through the years, but substantial amount of work and time has also gone to implement or improve other parts of Mapper.

In the next chapter reader is made acquainted with basic concepts of cartography, which are needed to understand following parts. Topics discussed include map projections and geodetic datums, different coordinate systems and nautical charts. An introduction to electronic nautical chart systems is also presented.

Chapter 3 begins with a brief history of Mapper, then the key points used in design and specification process are described. Choice criteria for development tools are also supplied as well as overview of similar products from other companies. Chapter ends with brief descriptions about all public functions of Mapper.

Chapters 4 and 5 are dedicated to implementation and usage phase of software, respectively. While this project is not very large or complicated, there still are a number of idiosyncrasies, where the problem hides behind smooth-looking surface.

Chapter 4 concentrates deeper in implementation phase of software development, revealing some of the intrinsic properties and decisions to the reader.

In the 5^{th} chapter the overview of existing applications which use Mapper is given. Currently Mapper is only used in different products of R-Süsteemid, such as hydrographic survey system software `PPHMS` and radar indicator panel `ASD`, but there are ideas for marketing it as a separate toolkit in the future.

The remaining problems and plans for future development are discussed in chapter 6.

In the appendixes there are found printed listings of two files from Mapper distribution. Appendix A contains public header file `mapper.h` where all functions, variables, constants and data structures, needed by user of Mapper library, are declared. Sample configuration file for Mapper, used with `PPHMS`, is listed in appendix B.

Chapter 2

Maps and Cartography

Up to now the process of map-making is progressed and nowadays precise maps together with several aiding mechanisms, such as satellite navigation systems and GIS, are used in almost every field of human activity. These trends require some elementary understanding of terms and techniques used in cartography from every user. The following chapter gives a short tour to help reader with these basic concepts.

2.1 Mapping and the Earth

From the old ages people have tried to visualize the shape of Earth. The beginning is indeed obscured and no one person or nation can really claim to be the first cartographer. Yet there are preserved samples of the maps of the world from Babylon dated back circa 600 B.C. [27].

Today everyone is convinced that the Earth is essentially a sphere. This simple fact faces us with quite serious problem when we want to display the surface of the Earth on two-dimensional plane, since one cannot map a sphere to a plane without distortion. Problem can be partially solved by using a three-dimensional model — called a *globe* — but this method is limited to certain scales (A globe with a diameter, say, 100 m has no practical value.). Besides, the two-dimensional media (like paper sheets or computer display) is much more comfortable to manage. Therefore the need for a transformation, mapping the surface of the Earth to a plane. Such transformation is called a *map projection*[1] [21].

Besides the projection there is a need for other information characterizing the data we will use to create a map. We need a *geodetic datum* — a reference system defining shape, size, position and orientation of a (mathematical) reference surface (e.g. sphere or spheroid). Also we have to find out which coordinate system (on the Earth) we are using.

[1]Some definitions of map projection require also inclusion of lines delineating meridians and parallels, which is usually carried out.

Beware that different countries may use the same parameters of the reference surface, but with different position and orientation! National Mapping Agencies may also use different map projections, based on the same reference system. Map projections depend on scale, area and distortion-characteristic needed. Exchanging map data between countries needs normally geodetic datum transformation, even if the same projection is used [25].

2.1.1 Coordinates

To identify the location of point on the Earth, one has to define a coordinate system. Although the use of Cartesian system is increasing with wider use of satellite positioning systems, coordinates of a a point are usually (and somewhat more naturally, for the Earth is nearly a sphere) expressed in spherical coordinates. In the latter case a net of (imaginary) longitude and latitude lines has been superimposed on the surface of the Earth, called meridians and parallels, respectively. This concept was worked out by Greek astronomer Hipparchus (2^{nd} century B.C.) and later refined by Ptolemy (2^{nd} century A.D.) [21, 27].

In order to define latitude and longitude it is necessary to define an equator and a zero meridian. The *Equator* is an imaginary plane perpendicular to the spin axis of the Earth and passing through the Earth's mass center. The *zero meridian* is an arbitrary reference plane which contains the Earth's spin axis. There is no natural choice for a meridian with longitude 0°, as the Equator is for latitude. Thus many meridians are referenced as prime meridians throughout the history, mostly on national basis. In 1884 the International Meridian Conference, held in Washington, agreed to adopt the

> "meridian passing through the center of the transit instrument at the observatory of Greenwich as the initial meridian for longitude,"

resolving that

> "from this meridian longitude shall be counted in two directions up to 180 degrees, east longitude being plus and west longitude minus" [21].

Since 1988 the International Earth Rotation Service (IERS) in Paris has defined the zero meridian (IRM, the IERS Reference Meridian) as the mean value of the longitudes of a number of observatories around the world. Also, the axis of rotation of the Earth is not fixed with respect to the solid mass of Earth but is in state of continuous motion (*polar motion*), affecting the precision of calculated coordinates. This effect, predicted by L. Euler in 1765, can move the pole (the point where the spin axis and the surface of the Earth intersect) between 5 and 10 meters per year. The IERS defines the IERS Reference Pole (IRP) for use in calculations [11].

There are three types of coordinate systems used — astronomical, geodetic and Cartesian coordinates.

Astronomical (Geographical) Coordinates

Astronomical coordinates form physically defined perpendicular coordinate system, based on the gravity.

The astronomical latitude ϕ_A of a point is defined as the angle between the vertical (the direction of the gravity vector) and the equatorial plane. By convention the Equator has the latitude of 0°, +90° (or 90° North) is the North Pole and −90° (90° South) the South Pole. Each degree of latitude is divided into 60 minutes and each minute in turn into 60 seconds of arc.

The astronomical longitude λ_A of a point is the angle between two planes, one of which is the local meridian (a plane passing through the vertical at the point and the spin axis of the Earth) and the other is the zero meridian (Figure 2.1.). Degrees of longitude are also divided into minutes and seconds. Unlike the degrees of latitude, degrees of longitude are of variable length, growing shorter towards poles.

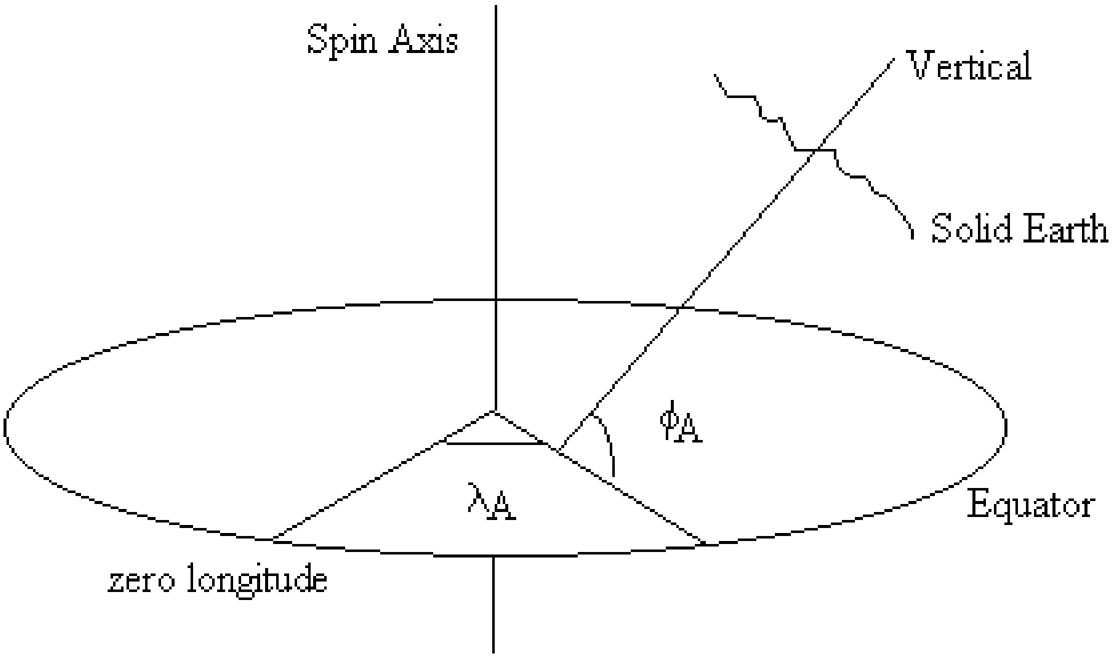

Figure 2.1: Astronomical latitude and longitude [11].

The astronomical coordinates are oversimplified and not suitable for precise mapping and navigation. They do not form a coordinate system in the geometrical sense, because if the directions of the vertical at two or more points on Earth are parallel, then these points will have identical (astronomical) coordinates.

Geodetic (Ellipsoidal) Coordinates

Geodetic coordinates form mathematically defined perpendicular coordinate system, based on the reference surface, specially ellipsoids, used for large and medium scale mapping, and in geodesy.

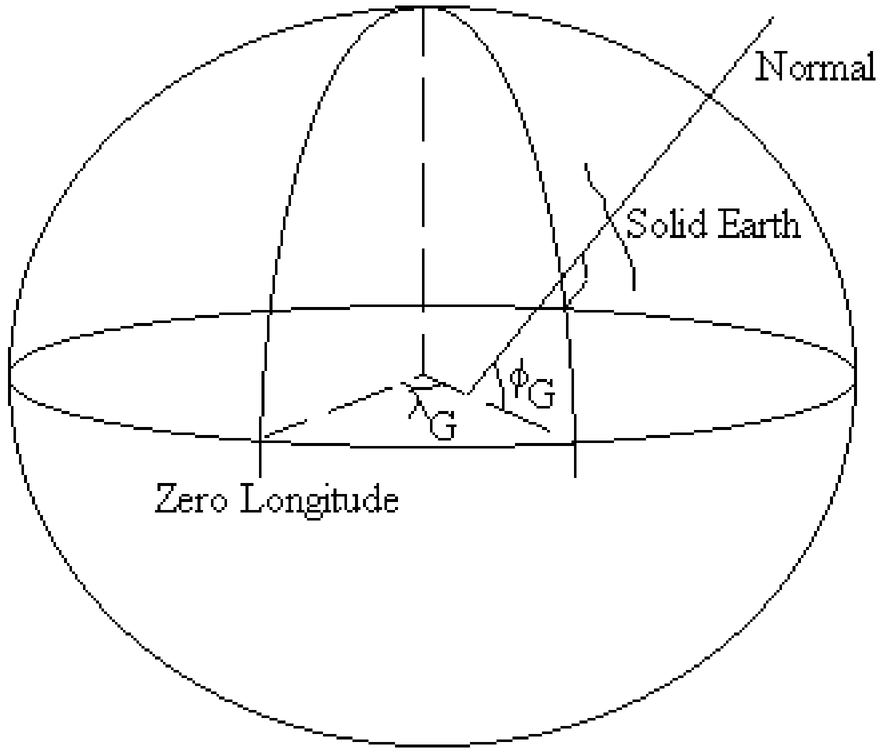

Figure 2.2: Geodetic latitude and longitude [11].

The position of a point on Earth in geodetic coordinates is defined by the ellipsoidal coordinates of the projection of this point onto surface of a reference ellipsoid along the normal to that ellipsoid (Figure 2.2.). *Geodetic latitude* ϕ_G is defined as the inclination of the normal to the ellipsoidal equatorial plane. Geodetic latitude is slightly greater in magnitude than corresponding astronomical latitude, except at the Equator and poles, where the two are equal. The geodetic meridian is defined as a plane through the normal and the minor axis of the reference ellipsoid and *geodetic longitude* λ_G as the angle between the meridianal plane of a point and the zero meridian (typically IRM).

Any two points on the Earth can have the same geodetic coordinates if and only if they lie on the same normal to the ellipsoid.

The third dimension — height — is usually given by the ellipsoidal height, measured along the normal. Typically the distance from geoid (height above mean sea level) has more practical value. Satellite positioning systems produce geometrical ellipsoidal height which is to be corrected according to the shape of geoid in current area.

Cartesian Coordinates

An alternative to spherical coordinates (astronomical or geodetic), where coordinates are expressed in angles, one can also measure a point's position on Earth's surface in Cartesian coordinate system along three perpendicular axes. These axes are usually labeled as X, Y and Z and form a left-handed triad (Figure 2.3.). The origin of the coordinate system may be either the center of associated reference ellipsoid or the mass center of Earth.

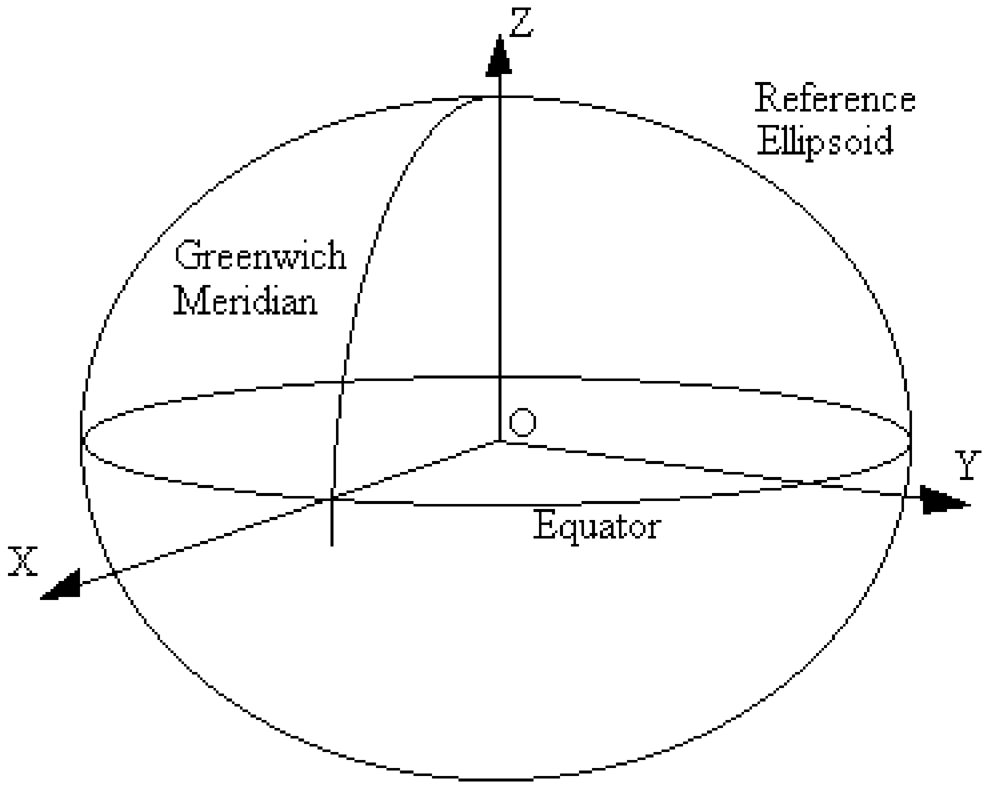

Figure 2.3: Cartesian coordinate system [11].

The greatest advantage of Cartesian coordinate system is the fact that they are completely defined by directions of the axes. Unfortunately, they are not so easy to use as spherical coordinates in many fields. For example, if a mariner sails the sea, he ignores the height and uses only latitude and longitude to mark vessel's position. In case of Cartesian coordinates all three coordinates change their values when moving and all are equally important in positioning. All satellite-based navigation systems (such as GPS) calculate coordinates of a point in Cartesian system and possibly later transform them to latitude and longitude [11].

Grid Systems

Rectangular, uniformly spaced grid is often used for user's convenience, as calculating latitude and longitude can become quite involved. Then coordinates of each point (sometimes called *cartographical* or *XY-coordinates*) are simply distances from two perpendicular axes on the flat map. The x coordinate is called *easting* (usually increasing east) and y coordinate *northing* (increasing north). Sometimes to these coordinates *false easting* and *false northing* have been added, respectively, to avoid negative coordinates. The grid lines usually do not coincide with meridians and parallels except for the central meridian and the Equator.

When the grid is superimposed on a map, it usually carries the name of the projection used for the map — that is, Transverse Mercator grid, Universal Transverse Mercator grid, etc. [15]. Grid systems are normally divided into zones to keep distortion within any zone below preset level. The boundaries between zones may be meridians of longitude (e.g. UTM) or determined by other means (state boundaries, ellipsoidal rhumb lines etc.) [21].

2.1.2 Geodetic Datums

Geodetic datums define the reference systems that describe the size and shape of the Earth. Hundreds of different datums have been used to frame position descriptions since the first estimates of the Earth's size were made by Aristotle.

Geographic coordinates may define different points on the Earth's surface, it depends on their reference system and *vice versa*, one point on the Earth's surface may have many geographic coordinates, depending on their reference system (geodetic datum) [25]. Referencing geodetic coordinates to the wrong datum can result in position errors of hundreds of meters. Different nations and agencies use different datums as the basis for coordinate systems used to identify positions in geographic information systems, precise positioning systems, and navigation systems. The diversity of datums in use today and the technological advancements that have made possible global positioning measurements with sub-meter accuracies requires careful datum selection and careful conversion between coordinates in different datums [3].

There are two principal types of datums: *vertical* and *horizontal*. Vertical datum is a level surface to which heights are referred. Horizontal datum is used as a reference for position.

The shape of the Earth is indeed spherical but more precisely it is like the shape of oblate ellipsoid (oblate spheroid), rotating around its shorter axis. The difference becomes significant on maps of scale 1:5 000 000 and it is absolutely necessary to take this fact into account when calculating coordinates for maps at scale 1:100 000 or larger [21].

To be more precise, the Earth is not an exact ellipsoid but *geoid* is the name given to the shape that the Earth would adopt if it were all measured at mean sea level. Due to irregular location of mass inside the Earth the geoid has quite complicated shape. The exact shape of the geoid is subject of continuous studies. The difference between well-fitting ellipsoid and geoid is rather small, no more than about a hundred meters (the difference between the ellipsoid and the sphere varies much more). Latitudes and longitudes as well as all planar coordinates are given with respect to ellipsoid, but heights and elevation lines are always reported relative to geoid.

The selection of constants defining the reference ellipsoid has been a major goal for geodesists since the 18^{th} century. These parameters are usually expressed by following constants (all four systems are directly interchangeable)

- semimajor and semiminor axes
- semimajor axis and flattening
- semimajor axis and eccentricity
- semimajor axis, geocentric gravitational constant and dynamical form factor (satellite-measured ellipsoids)

There are several reference ellipsoids which have been (and still are) used for producing maps. One can see from the table 2.1 on the next page that varying accuracy of measurements and gravity field irregularities result in quite different ellipsoid parameters. Up to space era ellipsoids were fitted to the Earth's shape (geoid) only over a particular country or region. The center of "local" ellipsoid (See figure 2.4 on page 11.) is not coincident with Earth's center of mass but the axes of reference ellipsoid are assumed to be parallel to the Earth's axes. New, satellite-determined, ellipsoids (e.g. WGS 84) represent the entire Earth more accurately but lack the "best fit" in any particular region.

Another important parameter in case of pre-satellite ellipsoids is "initial point", which is used to produce a datum. The coordinates of all control points are calculated relative to selected ellipsoid and initial point. For Earth-centered datums (used with ellipsoids, derived from adopted model of the Earth) the "initial point" is located at the center of the Earth.

Name	**Date**	**Equatorial radius**, m	**Polar radius**, m	**Flat-tening**	**Use**
WGS 84	1984	6,378,137	6,356,752.3	$\frac{1}{298.257}$	GPS, navigation
WGS 72	1972	6,378,135	6,356,750.5	$\frac{1}{298.26}$	NASA, oil companies
Australian	1965	6,378,160	6,356,774.7	$\frac{1}{298.25}$	Australia
Krassovsky	1940	6,378,245	6,356,863.0	$\frac{1}{298.3}$	Former Soviet Union
International Hayford	1924 1909	6,378,388	6,356,911.9	$\frac{1}{297}$	Elsewhere
Clarke	1880	6,378,249.1	6,356,514.9	$\frac{1}{293.46}$	Most of Africa, France
Clarke	1866	6,378,206.4	6,356,583.8	$\frac{1}{294.98}$	North America, Philippines
Airy	1830	6,377,563.4	6,356,256.9	$\frac{1}{299.32}$	Great Britain
Bessel	1841	6,377,397.2	6,356,079.0	$\frac{1}{299.15}$	Central Europe, Chile, Indonesia
Everest	1830	6,377,276.3	6,356,075.4	$\frac{1}{300.80}$	India, Burma, Pakistan, Afghanistan, Thailand, etc.

Table 2.1: Some official ellipsoids in use nowadays [21].

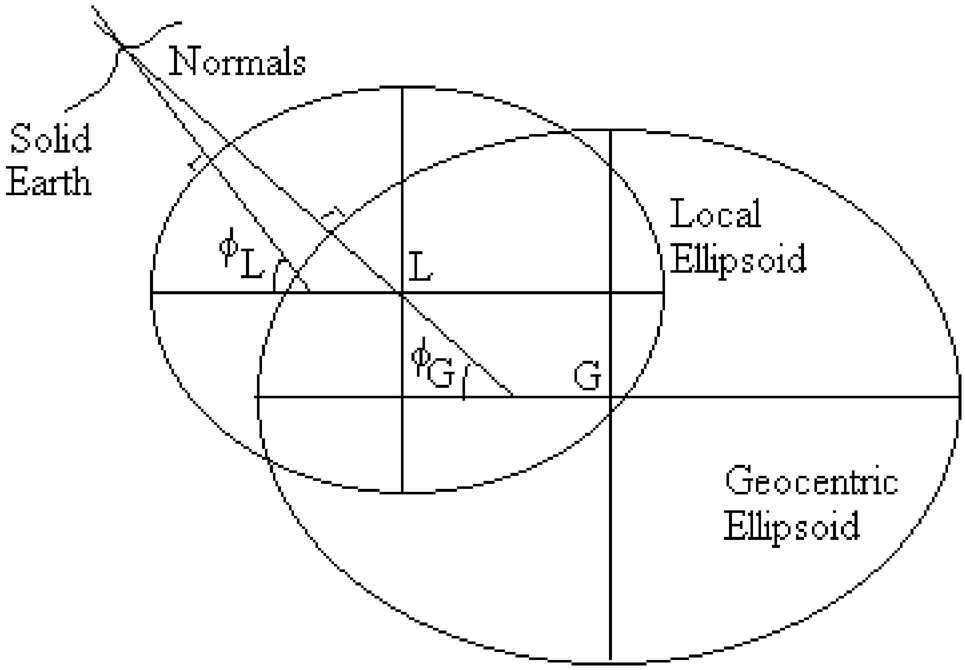

Figure 2.4: Local and geocentric ellipsoids [11].

WGS 84

The World Geodetic System (WGS), produced in 1984, is a geocentric reference datum, now internationally adopted as a standard for positioning and navigation. The measurements were initially held by United States Defense Mapping Agency.

The World Geodetic System provides the basic reference frame and geometric figure for the Earth, models the Earth gravimetrically, and provides the means for relating positions on various local geodetic systems to an Earth-centered, Earth-fixed coordinate system. The most accurate approach for obtaining WGS 84 coordinates is to acquire satellite tracking data at the site of interest and position it directly in WGS 84 using the Satellite Point Positioning technique. WGS 84 is also the reference system for GPS (NAVSTAR Global Positioning System) [26].

WGS 84 datum was defined using TRANSIT (a satellite navigation system similar to GPS) and therefore had only a limited accuracy (1–2 meters). During 1994 the Defense Mapping Agency redefined the basic coordinates and gravity field of WGS 84 using GPS, thus improving precision (less than 1 meter horizontally and less than 2 meters vertically) [11]. Another refinement was done in 1997, giving accuracy of coordinates of approximately 5 cm. Now the WGS 84 can be considered equivalent with IERS Terrestrial Reference Frame (ITRF) for all practical applications [12].

Datum Conversions

In order to combine coordinates based on different measurements in different systems it is necessary to understand the methods of transforming coordinates from one system to another.

There are developed a number of different procedures for coordinate conversion, each of them has its merits and drawbacks, which should be considered when choosing transformation. Differences of coordinates in two distinct datum may be expressed by nine parameters:

- the *semi-major axis* a and
- *flattening* (f) describe the shape of reference ellipsoid associated with each datum (The ellipsoid may be defined by other parameters, see page 9.)
- ΔX,
- ΔY and
- ΔZ express difference of origin between two ellipsoids
- θ_X,
- θ_Y and
- θ_Z are small rotations around appropriate axes to ensure that the axes in both systems are parallel.
- *scale factor* μ describes any differences in scale between two datums.

However very few transformations make use of all nine parameters.

Helmert transformation is a most general method of translating coordinates from one system to another. Cartesian coordinates are converted to another system using simple transformation matrix (ΔX, ΔY, ΔZ, θ_X, θ_Y, θ_Z and μ are involved).

If two datums of interest differ only by their origin (the center of the ellipsoid) then *Molodensky formulae* are suitable one-stage procedure.

Multiple regression formulae are used when there are regional changes of local scale and orientation, which are not taken into account by some simpler conversion method (e.g. Helmert transformation). The method involves the evaluation of a polynomial function (of X, Y and Z coordinates) at particular location. The coefficients for a polynomial are calculated using a best fit for a large number of points with known coordinates in both systems. Although simple, the accuracy of results is limited using this method [11, 3].

2.1.3 Map Projections

It cannot be said that there is one "best" projection for mapping. Actually there exist infinite number of different projections, using different parameters and giving different distortion in different areas of Earth. In real life only a relative small number of different projections are used, minimal distortion in given area (on large-scale maps mostly) is reached by varying several parameters, such as starting point or central meridian. Map projections are classified by several properties :

- **Area** — whether the projection is *equal-area* or not. On equal-area projections congruent bodies cover exactly the same area of the actual Earth when placed onto different parts of a map. Shapes, angles and scale must be distorted almost everywhere to achieve this property.
- **Shape** — a *conformal* projection has locally correct angles around almost every point on the map. Thus small areas have correct shapes while large areas are still distorted. Also maps with conformal projections have constant local scale and meridians and parallels intersect at right angles. No map can be equal-area and conformal at the same time. Many widely used projections are conformal.
- **Scale** — there are usually some lines with true scale, while no projection preserves scale throughout the map. *Equidistant* projections show true scale between one or two fixed points and every other point on the map.
- **Direction** — on *azimuthal* projections the directions of all points are shown correctly with respect to the center (there are also projections on which azimuths from two points are correct and on which directions from all points to one or two selected points are correct). One of these projections is also equal-area, another is conformal and another is equidistant.
- **Special characteristics** can be very appealing. For example, on the Mercator projection (See figure 2.5 on page 15.) all rhumb lines (lines of constant direction) are shown as straight lines, making it very comfortable for sailors to calculate the course. On the Gnomonic projection all great circle paths (shortest routes between points on a sphere) are shown as straight lines. Some newer projections are specially designed for satellite mapping.

Usually the Earth is projected to the developable surface[2] and then "unrolled" to the plane. Most map projections use cylinder (giving *cylindrical projections*), cone (*conical*) or plane (*azimuthal*) as an intermediate surface.

[2] A surface which can be transformed to a plane without distortion.

In case of cylindrical map projections a cylinder is wrapped around the Earth. If the cylinder touches the Equator throughout its circumference, the projection is called *regular cylindrical*. The meridians are projected as equally spaced straight lines, perpendicular to the Equator. The parallels are parallel to the Equator and mathematically spaced for certain characteristics. The best-known example of projections of this kind is the Mercator projection. If the cylinder is placed around some meridian instead of the Equator we get a *transverse cylindrical* projection, for example Transverse Mercator. If we put the cylinder around the Earth in a different way (i.e. its perimeter is not parallel to the Equator or some meridian) we have an *oblique cylindrical* projection. In the latter two cases most meridians and parallels are not straight lines.

If a cone is placed over the globe, with its peak along the polar axis of the Earth and the surface of the cone touching the globe along some particular parallel, a *regular conical* projection can be produced. The meridians are projected onto the cone as equidistant straight lines radiating from the apex. The parallels are shown as circular arcs centered on the apex. Lambert Conformal Conic projection is an example of conical projections.

A plane tangent to one of the Earth's poles forms the basis for *polar azimuthal* projections. All common tangent-plane projections of the sphere are azimuthal. The meridians are projected as straight lines radiating from a point but they are spaced at true angles (on conic projections the angles between meridians are smaller than true angles). Parallels are complete circles, centered on the pole. The example is Lambert Azimuthal Equal Area projection.

A projection can also be transverse or oblique conical (or azimuthal). In addition the cylinder or cone may cut the globe at two parallels instead of being tangent to just one, giving two standard parallels. Similarly, the plane may cut the Earth at any parallel instead of touching a point. Besides these general classes there are other projections, some of which are called *pseudocylindrical* (or *pseudoconic* or *pseudoazimuthal*) because they have only some properties of cylindrical projections (e.g. Sinusoidal) [21].

In following part a brief overview of some widely used map projections are given. More information, including actual constants and formulae can be found in [21] or other handbooks.

Mercator projection

The *Mercator projection* was apparently used by E. Etzlaub (1462–1532) of Nuremberg on a small map on the cover of some sundials, constructed in 1511 and 1513, but the principle remained obscure until G. Mercator (1512–94) independently developed and presented it in 1569 on a large world map (about 1.3 $\times$ 2 m). Mercator probably determined the spacing of parallels graphically since the tables of secants had not been invented.

E. Wright (ca. 1558–1615) developed later in England the mathematics behind the projection and in 1599 published tables of cumulative secants, indicating the spacing from the Equator [21].

Mercator projection (Figure 2.5.) is a conformal cylindrical map projection with the Equator as the standard parallel (The cylinder is wrapped around the Earth that way it touches every point of the Equator.). Meridians are equally spaced straight vertical lines but parallels, while still straight lines, are increasingly spaced toward each pole for conformality reason. Meridians cut parallels at right angles. The scale is true along the central parallel (the Equator) or along two parallels equidistant from the Equator. This projection is not perspective and its biggest drawback shows up with the poles, which lie at infinity, thus resulting (infinitely) great distortion of area in polar regions. This introduces so-called *Greenland problem* — Greenland appears to be larger than South America in Mercator projection (In reality South America has area eight times greater than Greenland.). This problem has caused Mercator projection to be less often used for world maps in published (especially educational) atlases.

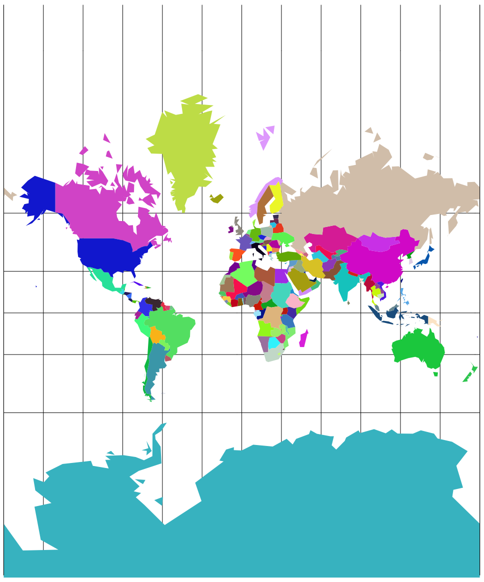

Figure 2.5: Mercator projection.

The major navigational feature of this projection is in the fact that a sailing route between any two points is shown as a straight line, if the direction of the ship remains constant with respect to north (sailing along rhumb line). Although the rhumb line (or loxodrome) is usually not a shortest way between two points (the great circle is), it is the way ships are traditionally navigated. Mercator projection has been widely used in navigation charts since its introduction and it has an important position even nowadays, especially for mapping equatorial regions [21].

Transverse Mercator Projection

The *Transverse Mercator* projection (TM) in a spherical form was introduced in 1772 by J. H. Lambert (1728–77) in his classic *Beiträge* with six other new projections. C. F. Gauss (1777–1855) analyzed the projection further in 1822 and L. Krüger published studies in 1912 and 1919 providing formulas for calculations, relative to the ellipsoid. Therefore the Transverse Mercator projection is sometimes called the *Gauss Conformal* or *Gauss-Krüger* projection. E. H. THOMPSON in 1945 and L. P. LEE in 1962 presented exact formulas permitting calculation of coordinates for the full ellipsoid.

Transverse Mercator projection is simply a Mercator projection where the central parallel is rotated 90°, thus providing a central meridian for the projection (The cylinder is wrapped around the Earth so that it touches the central meridian throughout its length.). The properties of this projection are also quite similar to regular Mercator — the Transverse Mercator is conformal, has true scale along the central meridian and the scale becomes infinite in areas which are 90° apart from the central meridian. The central meridian, each meridian 90° from the central meridian and the Equator are straight lines, all other meridians and parallels are complex curves.

This projection has caught the attention of cartographers only recently but in our times it is widely used, especially in its ellipsoidal form. Transverse Mercator projection is used for mapping large areas that are mainly north-south in extent. Distortion increases rapidly when distance from the central meridian increases, thus allowing precise mapping only in relatively narrow zones (10°–15° at most) [21].

Universal Transverse Mercator (UTM) is a rectangular grid system (first used by U. S. Army in 1947), applied to maps of the Earth's surface extending from the Equator to 84° N and 80° S latitudes. The UTM is a ellipsoidal Transverse Mercator with specific parameters applied — the surface of the Earth is divided into 60 numbered zones, 6° wide in longitude and 8° tall in latitude. Each quadrangle is further subdivided into 100 km $\times$ 100 km grid squares. A geographic location is given in meters as x and y coordinate pair, using the meridian between the two bounding meridians as a central meridian and reducing its scale to 0.9996 of true value. The lines of true scale are approximately 180 km east and west of the central meridian. Reference ellipsoid is chosen according to the particular region. The origin point lies on the Equator, at the central meridian and has an x coordinate of $500,000$ m and y coordinate of 0 in the Northern Hemisphere and $10,000,000$ m in the Southern Hemisphere [15, 21, 23].

Lambert Azimuthal Equal Area Projection

The *Lambert Azimuthal Equal-Area* projection (Figure 2.6 on the next page.) was introduced in 1772 by J. H. LAMBERT in his *Beiträge*. This projection is produced by projecting the Earth onto a plane, tangent to any point on a globe. Maps in Lambert Azimuthal Equal-Area projection feature equal-area but the projection is neither conformal, perspective nor equidistant. Scale decreases and distortion increases radially as the distance increases from the center point.

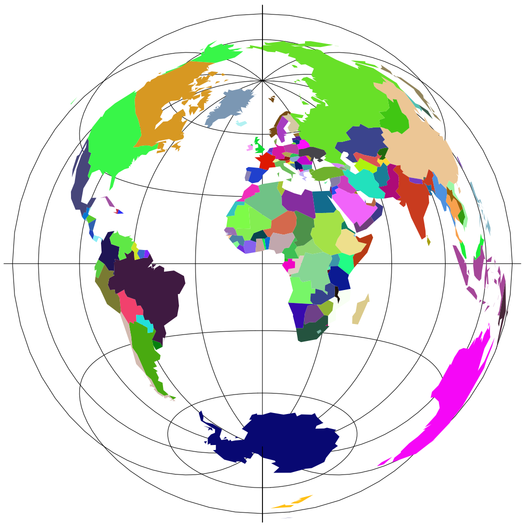

Figure 2.6: Lambert Azimuthal Equal-Area projection. Origin is at latitude 0° and longitude 0°.

All meridians in the polar aspect (plane touching a pole), the central meridian in other aspects, and the Equator in the equatorial aspect are straight lines. The outer meridian of a hemisphere in the equatorial aspect (for the sphere) and the parallels in the polar aspect (sphere or ellipsoid) are circles. All other meridians and parallels are complex curves. Directions from the center are true for the sphere and the polar ellipsoidal forms, directions from other points are never true. Point opposite the center is shown as a circle surrounding the map (for the sphere). Any straight line, drawn through the center point lies on a *great circle* (A circle on a sphere such that the plane containing the circle passes through the center of the sphere.).

The Lambert Azimuthal Equal-Area projection is best suited for regions extending in all directions from center point, such as Asia and Pacific Ocean [21, 15].

Albers Equal-Area Conic Projection

The *Albers Equal-Area Conic* projection (See figure 2.7 on the following page.) was presented by H. C. ALBERS in 1805. This conical projection is not conformal, equidistant or perspective but all areas on the Earth project proportionally to the map. Distances and scale are true only along two standard parallels (The parallels where the projecting cone cuts the Earth.). Directions are reasonably accurate in limited regions. Maps showing adjacent areas can be joined at their edges only if the maps have both the same standard parallels and the same scale.

This projection is well suited for large countries or other mainly east-west extending areas when equal-area representation is required [21, 15].

Projections of Estonian Maps

Estonian National Land Board and *Estonian Map Centre* are jointly producing two maps series, covering the whole area of Estonia. The *Estonian Base Map* is completed in its digital form, printed version is still under develop-

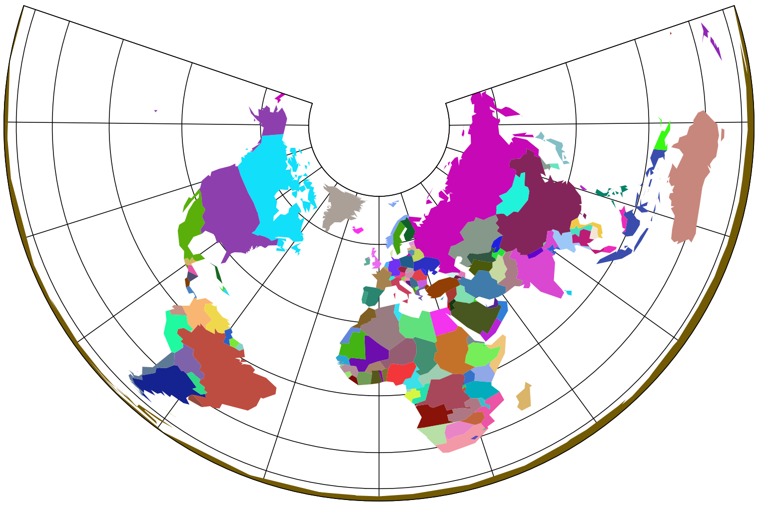

Figure 2.7: Albers Equal-Area Conic projection, standard parallels are 20° N and 60° N.

ment. This chart is based on satellite images, achieved resolution is 1:50 000. Transverse Mercator projection, with central meridian located at 24°00′ and scaled to 0.9996, is used for Base Map [24].

The *Estonian Basic Map* is expected to establish as a new standard for spatial data users in Estonia. Produced from aerophotos and printed on 550 sheets, it will have a scale of 1:20 000 (1:10 000 in digital form) [16]. Due to the shape and location of mapped area, Lambert Conformal Conic projection was chosen for this chart. This projection is conformal and has true directions in limited regions but area and shape are distorted away from standard parallels (58°00′ and 59°20′ in Estonia). The scale at the central parallel is 0.9999324284 (distortion 1/14800). For uniform distribution of the sheets of Base Map and Basic Map, false easting and northing are added to planar coordinates of every point [8]. Authorities also suggest Lambert Conformal Conic with given parameters for all mapping and geodetic works, carried out in Estonia.

The new nautical charts, issued by Estonian National Maritime Board in cooperation with *Regio* map publishing company, use Mercator projection and WGS 84 reference datum. Mercator projection is utilized in its secant form[3], central parallel is 60°. These maps are printed to paper in various scales and they also exist as digital charts (being in the focus for this paper) [14].

[3] In case of secant cylindrical projection the cylinder touches the sphere along two lines, both small circles (as opposed to great circle). The lines of intersection are treated as central parallels in computations.

2.2 Nautical Charts

Evidences are found that Polynesians had chart-like navigation aids long before Europeans reached the Pacific. They lashed together some sticks to describe prevailing winds and wave patterns [27]. For centuries, mariners sailing out of Europe (and later everywhere) have used paper maps for navigation purposes. Nautical charts were the major force behind the progress of cartography and geography at the early days of map-making. Up to the invention of typesetter by J. Gutenberg all maps were hand-printed and therefore very expensive and rare. In the era of great explorations most countries, sending out exploration ships (and also expeditions to inland areas), started to gather cartographers into special offices, the British Admiralty's being the most famous one. Today nearly every country has its own mapping agency, producing various kind of maps, usually only for areas inside country's borders[4]. According to SOLAS (*The International Convention for the Safety of Life at Sea*, originally adopted in 1914, following the *Titanic* disaster), all ships are required to carry "adequate and up-to-date charts".

At present time, SOLAS does not specify governmental responsibility for producing charts, but IMO (*International Maritime Organization*) has adopted resolutions, inviting governments to conduct hydrographic surveys and cooperate with other governments where necessary. Also, IMO members are urged to establish regional hydrographic commissions or charting groups and to support groups already set up by the *International Hydrographic Organization* (IHO) to prepare accurate charts (There are regions, not yet fully covered with hydrographical data of high precision.). An amendment to SOLAS is under development by IMO (likely to step into force in 2002), putting on contracting governments the onus to arrange for the collection and compilation of hydrographic data and the publication, dissemination and keeping up to date of all nautical information necessary for safe navigation, including official nautical charts, notices to mariners etc. [4].

Nautical charts (worldwide chart series — INT charts) are nowadays produced, using a single set of agreed specifications, set up by International Hydrographic Organization, with one nation producing a chart and all other nations wishing to cover the same area printing their charts from reproducibles, furnished by the producer nations. These guidelines, known as Nautical Chart Number One (INT 1) give strict color and symbol specifications, making charts of all agencies look alike. The programme for INT small (1:10 000 000 and 1:3 500 000) scale charts, developed by IHO in 1971, is completed. Despite the more complex problems associated with producing INT charts at large and medium scales, the essential standardization of symbols and format have been agreed and many of these larger scale charts have already been published [1].

[4]British Admiralty is the only organization producing nautical charts covering the whole world.

2.3 Electronic Maps

In modern computer era these machines have taken over much of the burden in the map-making craft and digital (electronic) maps, displayed from binary data at the request of viewer, are also rapidly moving into everyday life. Electronic charts have several advantages over traditional, paper-based maps. First, they are not tied to certain scale, user can choose any scale he wants, even the scale map was produced for is not the limit. Digital maps are not static, their appearance can change according to environment conditions and/or user interaction. On the other hand, digital maps are usually displayed on monitor, connected to some kind of computer. The size of viewable area of computer display peripherals is limited by technology, being considerably smaller today than the size of (navigational) charts, made of paper. The option of viewing large area is important in initial route planning, for example.

Such features make the idea of joining digital maps and data from some other database[5] very appealing. That way *Geographic Information Systems* (GIS) are formed — displaying data in several layers, some levels are filled with base maps (Earth's surface) and others with thematic maps, composed from database queries (e.g. roads, population density). GISes let users perform several queries and choose how the data is actually visualized. One special kind of Geographic Information Systems are *Electronic Chart Display and Navigation Systems* (ECDIS).

2.3.1 ECDIS

ECDIS is a GIS, used on board of ship for route planning and monitoring the position of vessel throughout the voyage. A certified ECDIS[6] is considered equal to and can be used instead of paper charts. Such a system must be capable of doing plenty of operations besides displaying a chart — automatic chart updates, entry of mariner's observations, route planning and showing current ship position, to name some [19]. An ECDIS must provide radar integration (chart is overlaid with radar image) and accurate ship location information (Two different redundant positioning systems are required, a differential GPS receiver being often used as a main source for current position.). There are several standards whose requirements have to be implemented — IMO/IHO Draft Performance Standard, IHO Special Publications S-57 and S-52 (see below) and others [2].

For aiding navigation, a simpler system can be used. An *Electronic Chart System* (ECS) does not need to meet strict standards as ECDIS. These systems use mostly raster charts[7] for map database and do not include all the

[5]Naturally, the other data set should be somehow geographically referenced.

[6]Certification is done by national shipping authorities and classification societies.

[7]A *raster chart* is basically just a scanned image of paper map, while in a *vector chart* each point on the chart is digitally mapped [4].

features of ECDIS (such as automatic updates or safe passage calculations). The Electronic Chart System can be used only for reference and all navigation have to be done with official paper charts from national Hydrographic Offices [2, 19].

The core of an ECDIS system is so-called *ECDIS kernel*, containing functions for displaying *System Electronic Nautical Charts* (SENC). SENC is common name for digital chart data, produced from ENC (*Electronic Nautical Chart*), issued by Hydrographic Offices. SENC is usually in ECDIS vendor's proprietary format, while ENC data must follow the IHO Special Publication S-57 "IHO Transfer Standard for Digital Hydrographic Data" (currently edition 3.0) [6].

The S-57 compliant map represents a specific kind of attributed vector data, consisting of *objects* (cartographic objects). In S-57 there are two kind of objects, *spatial objects*, holding information about geographical relations and *feature objects* which correspond to real world objects, distinguished by an *object class*[8] and a set of *attributes*, characterizing current object. Every feature object is linked with one or more spatial objects, depending on its *geometric primitive* (point, line or area), into tree-like structure. ENC charts in S-57 data exchange format (For edition 2.0 of S-57 it was called *DX-90* but currently the data exchange format is also named S-57.) are composed of such trees (*cells*), encapsulated, according to the ISO 8211 standard[9]. Besides data exchange guidelines, S-57 also includes *data dictionary*, a set of rules, specifying for feature objects, what combinations of classes and attributes are legal.

The S-57 specifies only the *situation model* of electronic chart, the way data is logically organized. There is nothing said about how to display this data, resulting in greater flexibility — the map database can be printed out as text or visualized graphically in several ways, using appropriate *presentation model*.

The presentation model for digital charts in ECDIS is specified by IHO Special Publication S-52 "Specifications for Chart Content and Display Aspects of ECDIS" (4^{th} edition) [5]. This document specifies the colors and symbols for all S-57 objects as well as requirements and suggestions for ECDIS display. S-52 is accompanied with Presentation Library, including machine readable definitions of symbol patterns, color tables (several color coding schemes are provided to match different light conditions on the bridge) and symbolization instruction look-up tables for binding Presentation Model to Situation Model (based on object's primitive, class name and attribute values). In digital Presentation Library every symbol has two distinctive

[8] A class is a common denominator for similar objects, e.g. all towers form class `TOWERS` and depth contours class `DEPCNT`.

[9] ISO 8211 is an *International Standardization Organization*'s standard for information interchange. It defines structured file format, readable by most computer systems in existence [22].

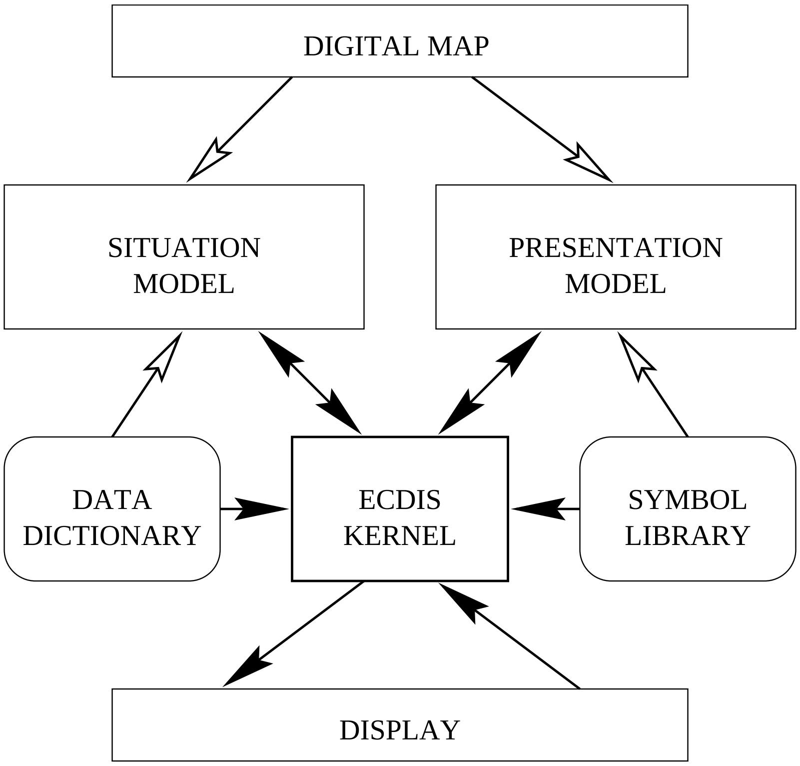

Figure 2.8: ENC display generator structure [13].

outlooks — *traditional* symbol set follows exactly the INT 1 standard, but the *simplified* symbol set is modified according to computer display target (symbols are simpler to draw and more colorful). The mariner has to have the ability to choose which one he likes best.

The task of ECDIS kernel is to combine map data (situation model) from SENC (ENC) and appropriate display primitives (presentation model) from IHO Presentation Library (Figure 2.8.). The actual image displayed on computer screen depends not only on chart contents but also on mariner's selections — map scale (Each object, conforming to S-57, has minimal and maximal visible scale attributes associated with it.), symbol set, color table and other settable parameters.

Chapter 3

Design

Mapper is a library of C-callable subroutines, giving to its user the ability to visualize on his X server nautical charts, conforming the IHO S-57 standard. The outlook of displayed maps is according to S-52 in most points. Mapper has no user interface of its own, all the interaction with user and other logic besides map display should be arranged by an application program, linked against the Mapper library (`libmapper.a` and corresponding header file `mapper.h`). The functions contained in this library are designed for maximal user convenience, letting him to build his programs just the way he likes. The interface should be general enough for making any kind of map viewers and editors. It should be noted that the library does not complain all the strict standards set up for ECDIS (this has not been the main goal so far) but is well suited for ECS and other non-critical uses.

Special care is taken to improve performance because there are large volumes of data in maps and there has been a need to use Mapper in time-critical environments.

3.1 Project History

The history of the Mapper project began in 1995 when the company *Virumaa Tiivad* made contract with ENMB to produce a software system for oceanographic survey ships (`PPHMS`, see section 5.1). The specification of this system included, among other things, requirements to display navigational charts for background information to hydrographer. After several prototype implementations a decision was made in 1996 to separate the source tree of map handling functions into separate project. Later the Mapper library has been funded by different contracts with ENMB, as well as from company's own resources. Version 1.1 was completed in the first half of 1996, with public interface almost the same as it is now. It featured displaying and editing (point objects only) maps, following IHO standards S-57 (edition 2.0) and S-52 in most parts. At that time development environment changed form

SCO OpenDesktop 3.0 to *Red Hat Linux 4.0* with *Metro-X 3.1* X-server.

From time to time several improvements were made (most of which were internal to library, affecting performance or accordance to mentioned standards), collected into version 1.2 in summer 1997. Also at that time all people, contributing to this project moved to new company, *R-Süsteemid*, which took over the development of Mapper. At the end of 1997, when there started to appear programs (and maps), following S-57 edition 3.0, modernization of Mapper became an actual question. Support for S-57 edition 3.0 (and appropriate version of S-52) is planned to version 2.0 of Mapper, which is under development at the time of writing (started in January 1998). Version 2.0 will be a total rewrite, resulting in better performance, code maintainability and better compliance to standards (visible to user especially in editing subroutines).

At present time Mapper is stable, relatively fast system for displaying and editing marine navigational charts. For financial reasons and simplicity it still does not obey the whole specification for nautical maps from International Hydrographic Organization (`PPHMS` would not benefit much from this fulfillment), but currently there is work in progress to teach Mapper understanding of maps from *ARC/INFO* GIS (especially Estonian Base Map).

3.2 Development Tools

As noted earlier, the need for map drawing routines raised in specification process of `PPHMS` and integration with the hydrographic survey software have indeed been one of the key points in Mapper. Thus the decisions made for `PPHMS` were essentially adopted for Mapper. Later, when they two were already separated, things were kept as they were for sake of simplicity and connectivity.

The tasks `PPHMS` is performing (See also [18] and [17].), automated data collection and visualization, put quite serious timing constraints to the software. The choice of *Unix* operating system was therefore quite straightforward — it has quite good capabilities for soft real-time processes, is widely spread in many variations and has a good reputation. And there is always subjective factors too, in this case the people who developed `PPHMS` (and later also Mapper) were familiar with Unix, resulting in shorter learning curve. As the budget of ENMB is limited, the `PPHMS` system was agreed to run on common IBM PC compatible personal computers[1]. As the PC platform supports very different (and powerful) hardware and software, this decision did not actually result in weaker performance or lost compatibility (Every step was carefully analyzed to provide maximum compatibility with other software and hardware systems.).

[1]The other hydrographic survey software systems run often on powerful workstations that give better floating point and graphics performance.

For the initial development platform *Santa Cruz Operation*'s SCO Open-Desktop 3.0 (Unix System V Release 3) was chosen, a system which included all necessary tools for development as well as for completed system's every-day operation. *X Window System, Version 11* was selected for graphical user interface and visual data representation (including nautical charts). X11 is network transparent window system that runs on very wide range hardware and is supported by many operating systems. A pure X11, as released by *The X Consortium*[2], is not very comfortable framework for graphical user interfaces, so an industry standard *OSF/Motif* library was used for that purpose.

Giving such development platform, picking a programming language was obvious. The *C* programming language is the language Unix was written in and surely it is the language with best supported development tools in any Unix box. X11 and Motif are also written in *C* and their functionality is best served if client program also uses *C*. The only alternative that should be probably considered if the project is started again today is *C++*. But in 1995 Motif did not have *C++* wrapper classes and there were no *Xlib* (the low-level interface to X11 in *C*) equivalent for *C++* that I was aware of. And for majority of PPHMS and Mapper development team, while skillful *C* programmers, *C++* is still more like magic. All other languages seemed to have very weak support from underlying software systems (Unix kernel system calls, X11) or were not fast or general enough.

At the beginning of 1997, *Linux* was voted for the new development platform for PPHMS and Mapper. Licensing costs of SCO, openness and more up to date tools available for Linux were the major arguments behind this decision. Linux with its freely available source code also proved advantageous when the time data spent in system buffers between coming in from serial port and reading by the application, became too long and uncertain. Little improvement to Linux kernel's serial driver gave a wonderful time-stamping resolution of 1 ms for PPHMS.

3.3 General Requirements

The general requirements for Mapper could be stated as two points — first, accordance to S-57 and S-52 from International Hydrographic Organization and second, easy and efficient cooperation with the PPHMS system. The first postulate, while short and clear, is not the easiest to complement. For example, as long as we have searched, none of the big and universal GIS systems (ARC/INFO, *Microstation* and others) can handle ENC data. The other reason (and not less important) for not choosing ready-made GIS engine was hidden in the second requirement. Main goal of PPHMS is data acquisition, the timeliness of data registering and saving is prior to any other

[2] In 1997 the development of X11 was transferred to *The Open Group*.

task, including chart display. With those expensive and general GIS engines it is usually hard (if not impossible) to gain any performance boost when necessary, a system, developed in-house is far more flexible. A general GIS system gives you good tools for adding your application to the main GIS engine, but the controversial task (to add GIS to your program) is usually far more difficult to achieve.

A proprietary library (miniature GIS engine) has solutions to all these problems — it would probably be cheaper (`PPHMS` is supposed to be installed on about ten survey vessels.), more integrated to main system and can be optimized when and where necessary (E.g. in Mapper there are not implemented some non-critical parts of mentioned IHO standards. As `PPHMS` is not intended to provide full ECDIS capabilities, certification is not required.).

The same everlasting "need for speed" has also interacted a lot with Mapper's internal data structures. One Mapper prototype used a free *PostgreSQL* client-server database system for data storage (Spatial data, even in vector form, occupies large amount of disk space. Digital data for area, equivalent to one sheet of paper chart is about 3–5 MB in size.). Forming a SQL query was indeed simpler than coding data storage routines on one's own but the loss of speed was even worse. So, the bottom line is, while using ready made parts and systems is good in general, in time-critical fields it is often necessary to do some additional work to get all requirements (especially for timing) satisfied.

3.3.1 Similar Products

Mapper is not the only product that can display maps in computer screen. Here follows a short and incomplete review of systems with similar capabilities, giving some context for Mapper's position.

EC2007

EC2007 [20] is an ECDIS kernel (a library of ECDIS functions compliant to the IHO standards S-52/S-57 and the IMO/IHO Draft Performance Standards), developed by *SevenCs Entwicklungsgesellschaft für Geo-Informatik mbH*, located in Hamburg, Germany. EC2007 is very similar (Both are function libraries which can be used for building up Geographical Information Systems.) to Mapper, running on wide variety of hard- and software (On *Sun*, *Hewlett Packard*, *DEC* and *Silicon Graphic* workstations and IBM PC compatibles with Linux, several flavors of Unix or *Windows NT* as an operating system.). EC2007 allows user to choose viewport, automatically update chart contents, display vector or raster data, query map data and add notes to chart. Also included are subroutines for route planning, managing sensor inputs, radar and *Vessel Traffic System* integration, and many others.

dKart base line

dKart base line [13] is a system from A/O Моринтех for creating and editing digital maps. Unlike the ECDIS system from SevenCs, this one is meant mostly for cartographers. This software package has great editing capabilities and can make use of different models of digitizers. Both data export to and import from charts in S-57 format is available in dKart base line. The dKart system works on *MS-DOS* operating system using proprietary graphical user interface toolkit *Window Driver*. This professional tool is used in ENMB to produce all nautical charts for Estonian waters.

Other Systems

Sure enough there are other systems invented for reaching similar goals which I am not aware of. In particular, there are two products, not meant for handling nautical maps, from which we are shamelessly stolen ideas to Mapper.

ivtools is a set of free *C++* frameworks for drawing and making map editors, developed by *Vectaport, Inc.* [7]. It is built around *InterViews 3.1* and *Unidraw* (graphics frameworks) by Silicon Graphics and *Stanford University*. There are also available several open source map viewers, editors and other GIS-related software at `http://www.vectaport.com/`.

Mayko Xmap is general map viewing application from *Mayr & Kockelhoff, Inc.* It has support for several vector and raster file formats (but not for S-57). In addition to simple navigation on the chart it has features for route planning, distances measurement and printing the map. If GPS receiver is present then Xmap will use it for accurate positioning [10]. The executable program of Xmap (and sample maps) are distributed free of charge. Both Xmap and ivtools run on Linux operating system (ivtools can be compiled under several Unix systems as well.).

3.4 Basic Data Flow

The main data flow for Mapper is sketched on figure 3.1 on the following page. Mapper forms separate layer between user application and lower level software systems like X and Unix. The program using map drawing functions need not worry about how the map files are read from the disk or how to draw chart symbol for particular building. All it has to do is open connections to X11 servers (There is arbitrary limit for supporting four simultaneous connections right now, this can be changed at compile time.) and all following low-level actions (e.g. allocating memory and other resources) are hidden behind Mapper interface.

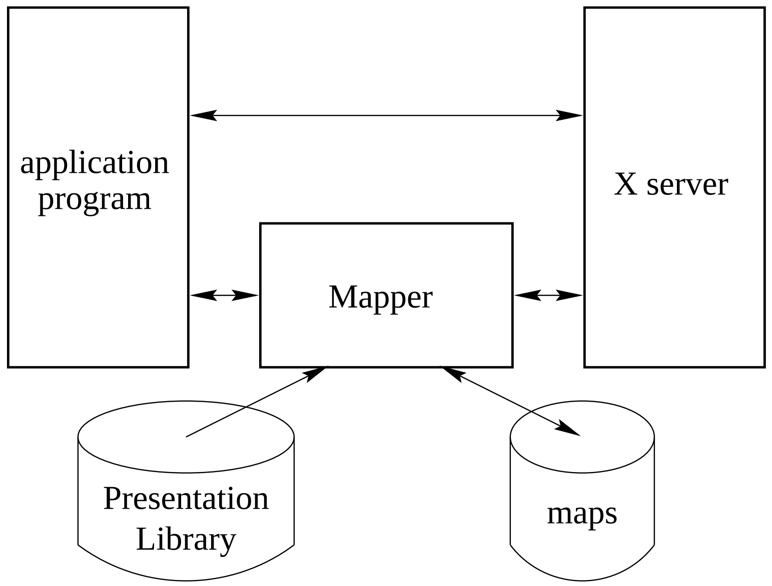

Figure 3.1: General data flow in applications using Mapper.

One of the basic ideas in Mapper design was encapsulation — application programmer need not (and must not) to peek into or change any of Mapper's internal data structures. In the *C* programming language this cannot be achieved as easily and elegantly as in native object-oriented languages, but the approach taken (using pointers to `void` in public interface functions instead of ordinary pointers to data structures) should be the simplest way to reach data encapsulation in *C*. Any information user should want to use from these structures is accessible via separate functions (like `MapGetMinX` or `MapObjGetType`). In addition to returning appropriate values these functions also check input parameters for legal values (see also page 48), increasing overall robustness.

A few data structures are still declared in public header file `mapper.h`. These are the very basic building blocks (`POINT`, `SOUNDING`, `SEGMENT`, `CONTOUR` and `ATTRIBUTE`), and are used to transfer data (possibly in large quantities) between user and Mapper. The `PPARAMS` structure is used for customizing screen appearance, different parameters are simply collected to this structure with two subroutines, `MapSetPresentationParams` to set and `MapGetPresentationParams` to query them.

3.4.1 Coordinate Transformations

Mapper uses its own file format for storing chart data on disk. Map files in native format are compiled from official charts (in S-57 format) with separate converter program `c3`, included in Mapper distribution. Such preprocessing boosts up map loading and saving significantly (Comparing map conversion from S-57 to internal format and loading data file in native format, the former process is several times slower.). Besides several other techniques done in map conversation stage, one with the most important rôle is calculation of coordinate projections for all points. This lets Mapper functions to deal only with projected coordinates and planar mathematics instead of spherical. Method has its drawbacks, though — the map data in internal format is tied to one projection, for changing it (or some projection parameter), the chart has to be reconverted from S-57 format.

The code for calculating projection transformations (and inverse transformations) is actually not the part of Mapper but forms a separate library. The application that uses services of Mapper is therefore required to link with both `libmapper.a` and `libproj.a`. The library includes code for Mercator and Transverse Mercator projections (due to requirements for `PPHMS`), which are calculated using power series (formulae provided by Snyder [21]), giving accuracy[3] better than 1 cm. Functions for setting map projection, its parameters and geodetic datum, and converting from geographical coordinates to cartographical (and *vice versa*) are provided and available to user.

Projected coordinates are held as 32-bit integer values, which are superior to floating point numbers in memory consumption (and also in performance in some microprocessors). User expects usually coordinates to be expressed in meters, but as data precision (after applying projection formulae) is often better, cutting off the fractional part would result in noticeable loss of accuracy. To overcome this problem, I decided to introduce a *multiplication factor* — data (in meters) is multiplied to this factor, giving an integral value (32 bits are sufficient for storing data about every point of the Earth with precision up to 1 cm.). User has to divide all coordinate values by the multiplication factor, if he wants them in meters (For example, 142.14 meters is encoded as 14214 with multiplication factor of 100.).

3.4.2 Handling Spatial Data

Spatial (geographically referenced) data can be represented as raster images (usually scanned paper maps) or true vector data, where every point is an entry in chart database. Mapper library can handle only fully vectorized datasets. Figure 3.2 on the next page displays the object model of Mapper. There are three important spatial data types in Mapper — representing a map as a whole (`MAP`), feature object[4] (`FOBJ`) and spatial objects (`SOBJ`).

Smallest cartographical entity is *object*, an elementary building stone, of

[3] Measured as distance from original point to the point, gotten by applying projection transformations and then inverse transformation formulae to the original point.

[4] Feature object is cartographical representation of a real-world object. It has all necessary parameters (attributes) but it does not hold any information about spatial placement, which is included in spatial objects.

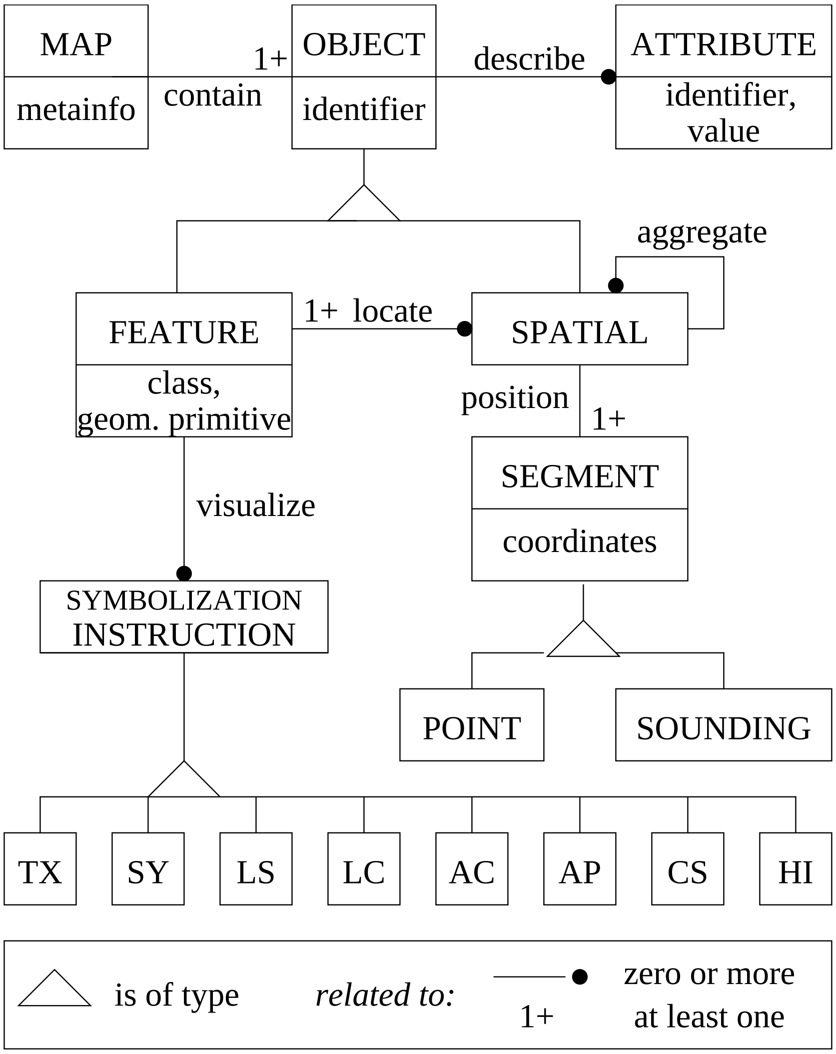

Figure 3.2: Data model of Mapper.

which all maps are composed. In Mapper, according to S-57, geographical data (locations of all points) are held in spatial objects, which can contain set of points, composed into an array of type `POINT` (forming one *segment* of line, corresponding to `SEGMENT` in `mapper.h`) and also links to other spatial objects, forming a tree structure. In addition to these downward links between spatial objects, each spatial object is also threaded to its parents, which may be either spatial or feature object. Spatial object can also hold depth data, if it is linked to feature object of type `SOUNDG`. Sounding points, as very important objects to mariner, have depth data along with coordinates in spatial objects, all other feature object classes have appropriate elevation information (if applicable) as part of their set of attributes.

Feature objects have almost one-to-one correspondence to real world objects[5], therefore needing more complex structure. First, every feature object has a *class* associated with it. A class (cartographic class) shows the type of object (IHO has standardized almost 200 different object classes.), for example, class `BUAARE` contains all areas with buildings. Feature object can also have one or more *attributes*. Attribute contains one property of that object, which can hold diverse values on different instances inside one class (class name is actually a very special attribute). S-57 defines all classes, attributes and their legal combinations.

Every feature object has also a *geometric primitive*, which can be either *point*, *line* or *area*.

- Point object has exactly one reference point (x and y coordinate) like pilot boarding places `PILBOP` or chimneys (IHO class `CHIMNY`).

- Line objects are made up from several points, forming either open or closed contour. Typical object with line primitive is coastline (`COALNE`).

- Areas are the most complex of the three. An area is defined by closed outline (exterior boundary) but is also allowed to contain holes (whose contours make up interior boundaries). In procedures `MapCreate`, `MapObjGetPoints` and `MapObjSetPoints`, these outlines are defined as of type `CONTOUR` and are simply collections of line segments. Object classes `DEPARE` (depth area) and vegetation area (`VEGARE`) are good examples of areas.

It should be noted that one object class may contain objects with different primitives (Land area — `LNDARE` — can appear as area or point, the latter representing very small piece of ground, only a bit larger than rock, which is shown on mid-scale charts as point.). Legal types for every class are given in S-57 [6]. Feature object has also pointers to appropriate symbology

[5] Some everyday objects, like buoys, are composed from several feature objects and there exist some feature objects which do not have a real-world equivalent at all (so called *cartographic objects*).

instructions, as one can see from figure 3.2. And last, but certainly not least, there are references to spatial objects in every feature object, holding the geographical points of that object.

A *map* in the context of Mapper is a collection of (both feature and spatial) objects, which can be referenced "as is" (i.e. sequentially) or (feature objects only) via an index by class names. There is yet another index, composed from object's class and attributes, used by chart display routines.

Application programmer who uses Mapper never sees data types that are not defined in public header file, but actually the structures used by functions with prefix `MapObj` are of internal type `FOBJ` (feature object). Similarly, `MAP` structure (actually pointer to it) is used everywhere the user documentation refers to a "map". Spatial objects are nowhere, even indirectly, visible to user, their contents is transformed to `SEGMENT`s and `CONTOUR`s.

3.4.3 Display Generator

As shown in figure 2.8, the main task of an ECDIS system is to provide seamless connection between situation and presentation models. Mapper has similar goal and although the majority of public functions deal with map data, rather than visualization issues, the latter is also important and is equally considered.

Every routine that deals with drawing has a parameter named *MapDisp*. This is a pointer to private structure `MAPDISP`, holding all necessary information about the visible part of a map. This structure contains a point in map to be centered and the extent of visible area (Width and height of the window.). In addition it holds scale information (meter-per-pixel values for both axes) and data necessary to issue X drawing requests. The functions with `MapDisp` prefix form an interface to the fields of this structure.

`MAPDISP` is the answer to "Where to draw?", more important question "What to draw?" is handled by the Presentation Library, worked out by International Hydrographic Organization [5]. Presentation Library is machine-readable data that provides shapes for chart symbols, colors and look-up tables for matching S-57 data (feature object's class and attributes) with appropriate presentation by S-52.

The Presentation Library files, compiled by IHO, are preprocessed for Mapper system by separate utility program to make them more compact and to do beforehand as many calculations as possible. The processed files are then read in by Mapper during the initialization phase. At run time the appropriate piece of map is drawn to specified pixmap using the symbolization instructions, looked up from given tables (more on this topic in section 4.5).

The presentation side of Mapper was designed with portability and extensibility in mind, based on widely accepted standards (S-52, X Window System technology), which help to make this software very flexible. X11 is a network transparent windowing system and provides very powerful tools for making client-server type applications also with Mapper — maps and the application itself can reside on some high-end server but actual images are provided to workstations with good graphical capabilities. At present time there is a limit of four simultaneous open displays (it can be easily changed at compilation time), which can display similar or different maps at the same time, depending on application needs[6].

The situation and presentation models are joined together with special structure called `METADATA`. It is designed to allow different combinations of data holding and drawing architectures to be used, but up to now only S-57 and S-52 are implemented. `METADATA` holds the key parameters of every model (Look-up tables, color and symbol definitions, etc. for S-52, lists of all class and attribute names for S-57.). Every instance of Mapper can use only one combination of situation and presentation models, given as parameters to `MapInit` (Of course there can be more than one separate process using Mapper code with different parameters/models at a time.). There was no idea about incorporating different ideologies at the time of original specification, therefore the public interface is somewhat inconsistent now. Future plans include clear separation of the model-specific and general functions, hopefully making the library even easier to use.

3.5 Functional Decomposition

3.5.1 Public and Private Functions

An important decision for all systems is the choice of user interface. When designing Mapper's interface, a decision was made to keep all important data structures internal to library. This approach caused among other effects also increased number of public functions — instead of modifying some variable directly, programmer must now use special function to change the value of the variable (and probably another routine for querying it).

Besides the public interface there are many more procedures, which are used only internally by Mapper. Actually, many functions of common usage are only wrappers around some other subroutine which does the job. There are two reasons for such duplication. First, prototypes for inner and outer functions are different, private functions use "real" data structure names (e.g. `FOBJ`), while public ones use pointers to `void` instead. Second reason has its roots in error trapping — all procedures described in `mapper.h` need to have some kind of error detection built-in for user-entered data. Internal functions do not need to look out for exceptions so much, gaining more speed instead. Private functions, while not discussed here in depth, form the core of Mapper and are responsible for most duties.

[6] On every display there can be any number of windows with arbitrary number of different charts opened at the same time.

All public entrance points to Mapper (having all the `Map` prefix) are shown in table 3.1.

Full *C* language prototypes can be found in appendix A. As one can see from the table, all 62 subroutines can be divided into four sections. Rest of this section explains the ideas and usage of these functions in more detail.

Section	Function Name	Description
General purpose subroutines	`MapInit`	Initializes Mapper library
	`MapVersion`	Returns version of Mapper
	`MapGetClassNames`	Gives class names
	`MapErrorString`	Description of last error
Functions for handling maps	`MapCreate`	Creates an empty map
	`MapLoad`	Loads a map into memory
	`MapUnLoad`	Frees map contents
	`MapDestroy`	Deallocates a map
	`MapSave`	Saves map to external file
	`MapGetMinX`	Returns minimal x coordinate
	`MapGetMinY`	Returns minimal y coordinate
	`MapGetMaxX`	Returns maximal x coordinate
	`MapGetMaxY`	Returns maximal y coordinate
	`MapGetFirstObject`	Returns first object in map
	`MapGetNextObject`	Gets next objects sequentially
	`MapGetFirstObjectInClass`	Returns first object in class
	`MapGetNextObjectInClass`	Gets objects of one class
	`MapGetObjByCoord`	Finds object by its location
	`MapGetCoordinateMagnification`	Asks for magnification factor
	`MapSetCoordinateMagnification`	Specifies magnification factor
	`MapGetSoundingMagnification`	Gets magnifier for soundings
	`MapSetSoundingMagnification`	Sets amplification for soundings
	`MapGetProjection`	Projection currently used
	`MapGetHorDatum`	Returns geodetic datum used
Object handling procedures	`MapObjGetMinX`	Returns minimal x coordinate
	`MapObjGetMinY`	Returns minimal y coordinate
	`MapObjGetMaxX`	Returns maximal x coordinate
	`MapObjGetMaxY`	Returns maximal y coordinate
	`MapObjGetAttributeCount`	Returns number of attributes
	`MapObjGetAttributeValue`	Gets value of specified attribute
	`MapObjSetAttributeValue`	Sets value of specified attribute
	`MapObjGetAttributes`	Gets values of all attributes
	`MapObjSetAttributes`	Sets all attributes at a time
	`MapObjDelAttribute`	Wipes out specified attribute
	`MapObjDelAttributes`	Deletes all attributes
	`MapObjGetClass`	Returns class name of the object

Continued on next page

Continued from previous page

Section	Function Name	Description
	`MapObjSetClass`	Changes object's class
	`MapObjGetType`	Returns geometric primitive
	`MapObjGetPointCount`	Gets number of points in object
	`MapObjGetPoints`	Returns objects' geometry points
	`MapObjSetPoints`	Sets points of the object
	`MapObjHighlight`	Marks object with a visible cue
	`MapObjCreate`	Makes a new map object
	`MapObjDestroy`	Discards existing object
Display manipulation	`MapDraw`	Draws entire map
	`MapAreaDraw`	Draws specified area only
	`MapObjDraw`	Draws selected object only
	`MapObjAreaDraw`	Draws object in given area
	`MapSetPixPerM`	Specifies display pixel size
	`MapDispCreate`	Allocates `MAPDISP` variable
	`MapDispSet`	Specifies drawable area
	`MapDispDestroy`	Frees `MAPDISP` memory
	`MapGetPresentationParams`	Queries presentation elements
	`MapSetPresentationParams`	Changes aspects of drawing
	`MapGetColor`	Provides one color value
	`MapGetColors`	Gives values for all colors
	`MapGetColorScheme`	Returns active color scheme
	`MapSetColorScheme`	Changes color scheme
	`MapCntl`	Toggles map's visibility
	`MapClassCntl`	Changes visibility of a class

Table 3.1: Public interface of Mapper library.

3.5.2 General Purpose Functions

All four members of this group perform quite general tasks that did not fit well into other categories.

`MapInit` is the only prominent function that is absolutely required to complete successfully before any other subroutine from Mapper can do its work. `MapInit` carries out several initialization tasks like configuration file loading and parsing, Presentation Library loading, memory allocation for buffers and pixmaps and so on. User have to give several parameters to `MapInit` — open connections to X server, situation and presentation models to be used in this session (Only S-57 and S-52 are currently supported.) and Mapper configuration file name. This file contains descriptions of map classes and colors for all color schemes as well as some other parameters. Sample configuration file is provided in appendix B.

MapVersion returns current version of Mapper. The major version number is in bits 16..31 and minor revision in bits 0..15 of return value. Currently, for version 2.0 of Mapper, MapVersion returns *0x20000*. Procedure MapGetClassNames returns names and short descriptions of all map classes that Mapper knows about (i.e. that are found in configuration file). Memory is allocated for returned character strings and application should free it if these strings are no longer needed. The last function in this group, MapErrorString, can be used for error tracking. If some function cannot complete its work successfully, return value is set to indicate an error (-1 or NULL usually). In this case MapErrorString returns a pointer to message, describing the abnormal situation (User should never attempt to deallocate this string.). A public variable MapError holds numerical error code[7], similar to errno/perror approach used by UNIX system calls. MapError (and output of MapErrorString) is *not* reset when a procedure returns successfully.

3.5.3 Map Manipulation

These functions are meant for activities, carried out with entire map. Entries in this section cover the topics of reading and writing maps to disks, asking for and changing chart's properties as well as searching for objects.

MapCreate is a function that have to be called before any other action with a chart. It returns an empty MAP structure, which is usually passed to MapLoad for filling with map data. Alternatively, user can add objects to the bare map to make a chart of his own. MapUnLoad is a fellow of MapLoad that "undoes" the loading process, after executing MapUnLoad the MAP variable is ready for loading another data set. When a map is no more needed, a call to MapDestroy should be issued. This subroutine (a complement to MapCreate) wipes out map's contents as well as MAP structure, which should never referenced again. There is no need to call MapUnLoad before MapDestroy. Any changes made to a chart can be made permanent by calling MapSave, which writes the data from memory to disk.

Following four procedures return the outermost coordinates (in planar coordinate system (grid) with specified projection and parameters) of particular chart. MapGetMinX, MapGetMinY, MapGetMaxX and MapGetMaxY return the minimal and maximal coordinates along the horizontal and vertical axes, respectively. Locations of all points belonging to that map reside inside the rectangle, defined by these four coordinates.

The next group of procedures is provided for retrieving single objects from map data pool. MapGetFirstObject and MapGetNextObject are useful for "walking through" all objects. MapGetFirsObject should be executed first (it sets up several inner counters), following one or more calls to MapGetNextObject. Both functions return NULL if there are no more objects.

[7] Values for MapError are defined in ERRCODE enumeration type in mapper.h.

This makes error determination a bit more complicated (As `NULL` marks also abnormal termination.) — value of `MapError` should be cleared before calling these functions (and the following pair). `MapGetFirstObjectInClass` and `MapGetNextObjectInClass` are very similar to `MapGetFirstObject` and `MapGetNextObject` in all aspects except they perform their work only among objects of specified class. There is also possibility to search for an object by its coordinates. `MapGetObjByCoord` returns all objects that are situated near given point (User can specify how far is still "near" by special parameter to this procedure.).

Following functions give to user control over several parameters, associated with every map. `MapGetCoordinateMagnification` lets user to investigate coordinate magnification factor[8]. The value of this parameter is editable via `MapSetCoordinateMagnification` procedure. There are resembling subroutines for asking and setting a multiplication factor, used to scale the sounding (depth point) values (`MapGetSoundingMagnification` and `MapSetSoundingMagnification`).

`MapGetProjection` gives information about the projection and its parameters currently used. Map projection is coded as an integer number in accordance to S-57 [6]. The standard specifies 14 different map projections but Mapper can currently apply only Mercator and Transverse Mercator. `MapGetHorDatum` returns geodetic datum used. As with projection, there are 228 coded datums specified by International Hydrographic Organization, but in Mapper there is now support for WGS 84 and Krassovsky datums only. Mapper holds coordinates in data files, projected to plane, so the only way for changing map projection or geodetic datum is to convert Mapper chart files to S-57 format (which contains geographical coordinates) and back with another projection.

3.5.4 Objects from Every Aspect

Subroutines belonging to this group provide an interface to individual map objects. Objects[9] are very basic "bricks", of which map is composed. Usually one object corresponds to one object from real world, but there are several exceptions.

`MapObjCreate` is used to make a new instance of some object class. The object is also added to the database of given map. `MapObjDestroy` deletes object from map and removes it from memory.

A special function `MapObjHighlight` is used to visually differentiate object from anothers. Every point belonging to that object is surrounded by orange ring and the whole area is shaded if object's geometric primitive is

[8] As Mapper holds coordinates as integers, the actual coordinate value (in meters) is calculated by dividing the value returned by Mapper by the multiplication factor.

[9] In this context, *object* means actually *feature object*.

area. Stress is removed after calling this procedure again with parameter *Highlight* set to 0.

Following four subroutines — `MapObjGetMinX` and `MapObjGetMaxX` along horizontal axis, and `MapObjGetMinY` and `MapObjGetMaxY` for y (vertical) axis, return object's outermost coordinates. For point object, `MapObjGetMinX` equals to `MapObjGetMaxX` and `MapObjGetMinY` comes up to `MapObjGetMaxY`.

Every object belongs to exactly one class, settable by `MapObjSetClass`. Class name is a string of at most six characters (constant `CLASS_NAME_LEN` is defined in `mapper.h`). Assigned class name can be retrieved by calling `MapObjGetClass` function. Every object has also a geometric primitive associated with it. Primitive type of current object (point, line or area) is returned by `MapObjGetType`. Geometric primitive is not applicable for changing after object creation.

Object can have one or more attributes, describing the object in more detail. Every attribute has a name and value (character string or integer) as described in `ATTRIBUTE` structure in `mapper.h` header file. Convenience function `MapObjGetAttributeCount` returns the number of attributes belonging to given object. Value of single attribute can be queried using `MapObjGetAttributeValue` and modified with `MapObjSetAttributeValue` subroutines. `MapObjDelAttribute` is useful when there raises a need to remove some entry from object's set of attributes. Control over the full set of attributes, belonging to one object, is achieved with `MapObjGetAttributes`, `MapObjSetAttributes` and `MapObjDelAttributes`. Unlike previously mentioned `MapObjGetAttributeValue` procedure, `MapObjGetAttributes` gives back copy of actual attributes, which should be freed by caller.

Number of points in an object are returned by `MapObjGetPointCount`. Coordinates of points can be asked by calling `MapObjGetPoints`. Returned coordinates are grouped into segments and contours (exterior and interior). Changed coordinates are updated with `MapObjSetPoints`.

3.5.5 Keeping Display Sane

Most subroutines in Mapper deal directly or indirectly with data visualization. One group of functions is dedicated purely to display manipulation — setting scale and visible area of map, changing several parameters affecting visual representation of charts and of course subroutines that produce the actual image. Most used services are probably `MapDraw` and `MapAreaDraw`, for drawing map data full-screen or to specified portion of window, respectively. There are couple of functions for drawing a single object as well, named `MapObjDraw` and `MapObjAreaDraw`. All of these use data from specified map to draw into *Drawable*[10], specified in *MapDisp* parameter. Drawing process

[10] Drawable is a data type in X with capabilities to receive input from drawing functions, either a *window* or *pixmap*.

uses *MapDisp* to determine what to draw and also uses preset color scheme and other parameters from appropriate `METADATA` structure.

Viewport and scale changes are done via calls to `MapDispSet`. The variable of type "pointer to `MAPDISP`" should have been set up by `MapDispCreate`. The memory storage, allocated by `MapDispCreate`, have to be freed, using `MapDispDestroy`, when it is no more referenced. The logic behind the coordinate transformation for display is opened on page 45. Before calling any drawing function it is highly recommended to calibrate Mapper for current monitor. For `MapSetPixPerM` user needs to specify two parameters, how many pixels are there in line with length of one meter in directions of both axes.

The answer to "What to draw?" is addressed by following two procedures. `MapCntl` toggles on and off drawing of the whole chart (all classes), while `MapClassCntl` controls the visibility of individual classes. Initially all classes are excluded from display list.

Current color coding scheme can be queried or changed using functions `MapGetColorScheme` and `MapSetColorScheme` and constants from `mapper.h` file (`CCS_DAY_BRIGHT` and others). Color coding scheme is a table of color values (Red-Green-Blue model), different color schemes are provided for changing light conditions on bridge. Other parameters, affecting generation of chart display are controllable by `MapGetPresentationParams` and `MapSetPresentationParams`. This pair takes for arguments a pointer to `PPARAMS` structure (which is defined in public `mapper.h` file), and an unsigned long integer, containing one or more flags, logically added (*OR*ed) together. These flags (`PP_TWO_SHADES` and like) are defined also in the same header file.

For user convenience, the `MapGetColor` and `MapGetColors` routines are presented. They return a *Pixel*[11] definitions of one (`MapGetColor`) or all colors (`MapGetColors`) in current scheme. These colors can be used, for example, in application's user interface[12].

3.6 Application Framework

An application indeed needs some general structure to adopt the Mapper library flawlessly into the system. Functions, essential to any user program which uses Mapper's services, are shown in figure 3.3 on the next page.

First, the Mapper have to be initialized (supposedly when application is starting) via a call to `MapInit`. After `MapInit` has successfully returned, a map drawing system is properly set up, according to initialization file contents and other parameters to `MapInit`. `MapCreate`, `MapDispCreate`

[11] Pixel is an X11 data type, representing index into the *colormap*.

[12] S-52 from IHO gives strict guidelines for using colors in user interface. All necessary color definitions are included in Mapper color tables.

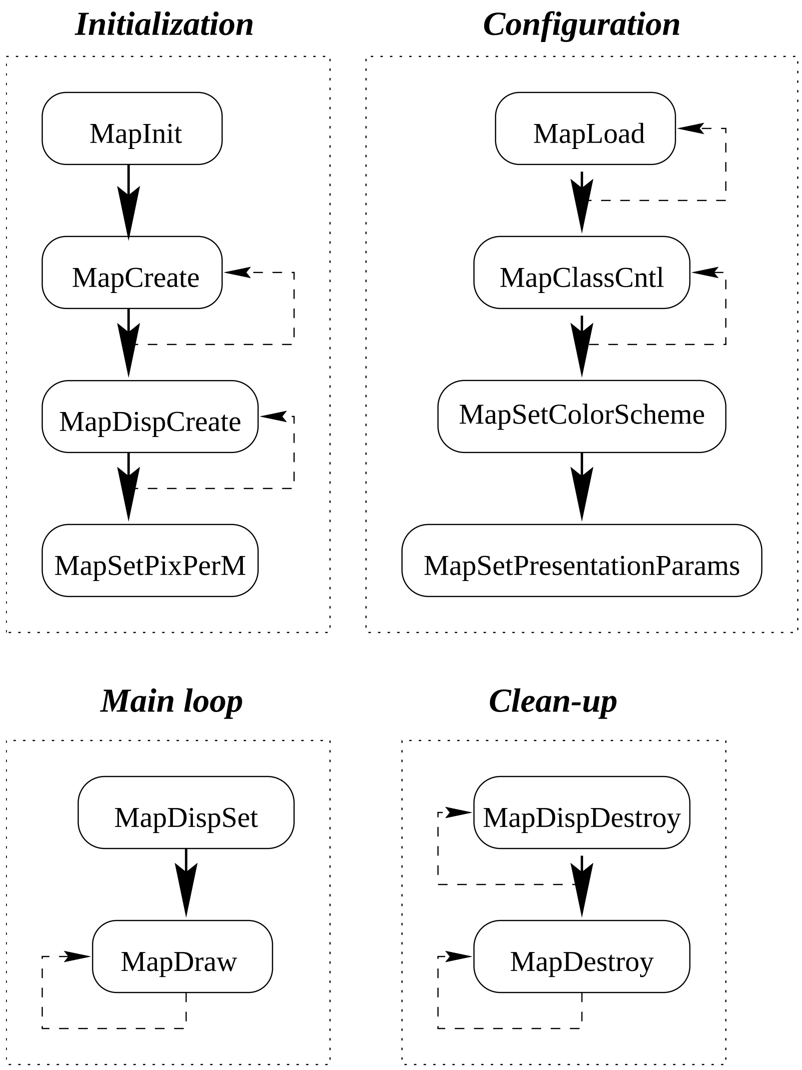

Figure 3.3: A skeleton of very simple map viewer.

and MapSetPixPerM are usually called before main part of the program. MapCreate allocates memory for a MAP structure. Such a structure is needed by all other chart handling functions (load, save, draw etc.). If there is a need for more than one map at a time then MapCreate should be executed more than once. A call to MapSetPixPerM calibrates Mapper to current display size (Or whatever size parameters are passed to this function by user.). MapDispCreate returns a pointer to MAPDISP which is used by map-drawing subroutines to determine scale and visible area. Separate variables of type MAPDISP * (actually for user they are seen as of type void *) are needed for two windows with different position or scale (Even if the same map is displayed in both windows.).

The main loop of a program usually includes one or more calls to MapLoad for fetching the chart data from disk. Prior to loading another chart to same MAP structure, MapUnLoad have to be executed. When data is loaded into memory, user wants to call MapClassCntl once for every cartographic class he wishes to be visible. If he chooses to change some default visualization parameters, MapSetColorScheme and/or MapSetPresentationParams can be called. The "heart" of Mapper — MapDraw should be called every time screen update is needed. When the visible area is changed, MapDispSet is needed before MapDraw.

Before the application finishes, calls for MapDestroy and MapDispDestroy should be issued. These procedures free memory allocated by MapCreate and MapDispCreate, respectively, and should be issued once for every instance of MAP and MAPDISP.

These are only rough guidelines, in real application more calls to different functions from Mapper library are usually needed, but the overall structure should still remain the same.

Chapter 4

Implementation

4.1 General Principles

Current chapter takes a quick look "inside", covering some more important and original concepts of implementation of Mapper.

In Mapper development one of the most important aspect has been the code reusability and compatibility. With 16,590 lines of source code (Version 1.2, a still-in-development version 2.0 has 14,799 lines up to now, probably increasing by 15–30 per cent when completed.) and several man-years of work, the clean and unencumbered source code is one of the keys to success.

Code reuse is achieved, besides harmonizing the inner code base, by using special library that is common to all projects of the R-Süsteemid company. This library, `rslib`, contains some data structures and appropriate procedures to use these data types.

Mapper uses three data types from the `rslib` library — general hash tables, pointer arrays (`PA`) and generalized array (`SA`). The package providing hash table management is a slightly modified version of J. Coffin's public domain software, other two are developed locally. `PA` is a self-growing array of general pointers, I found it very useful for storing several indices that Mapper needs to build and store (e.g. list of objects to display, objects sorted by class name, etc.). Generic array `SA` forms array of arbitrary sized elements what comes handy when data is to be read from files in continuous stream (like charts). Hash tables are used for implementing look-up tables for matching feature objects and their presentation (symbology instructions).

4.2 Source Tree Organization

Mapper source code is dispersed between several files, grouped to directories for better navigation, forming branching source tree.

```
include/
mapper/
    presentation/
    data/
    io/
    utils/
projlib/
    include/
    src/
conv/
    conv.2/
    conv.3/
test/
```

This tree is relative to `/home/mercator/`, current home directory for this project. Subdirectory `include/` holds several header files, needed for this project, in `mapper/` is the principal material for Mapper, containing sources for visualization (`mapper/presentation/`), spatial data manipulation (`mapper/data/`), chart input and output procedures (`mapper/io/`) and in `mapper/utils/` various utilities that are part of Mapper (e.g. initialization and error reporting functions). Under `projlib/` there are functions for doing projection transformations and several other common subroutines. The contents of this directory is compiled into separate library. `conv/` is reserved for file format converters. `conv/conv.2/` holds source code for transforming files between DX-90 (obsolete) and Mapper formats, converter for S-57 version 3.0 lives in `conv/conv.3/`. Subdirectory `test/` contains source code for `mtest` program which is used for in-house testing, being described in more detail on page 55.

The binary distribution of Mapper is set up as following. All path names are relative to the base, which is determined from `MAPPER` environment variable. This environment (shell) variable is used for locating the files of Presentation Library and also map data files if they do not have full path specified. Convenient place where `$MAPPER` points in Linux system is `/usr/local/mapper/`.

```
bin/
include/
lib/
plib/
maps/
    ...
smaps/
    ...
doc/
```

In this hierarchy, `bin/` directory holds several executable programs that accompany the distribution, like chart and Presentation Library converters. `include/` is a place for public header files (`mapper.h`), in `lib/` there is the actual library — `libmapper.a`[1]. `plib/` is reserved for Presentation Library files (fill patterns, look-up tables, etc.). `maps/` is convenient place for holding maps. In subdirectories of `smaps/` one could want to keep the original maps from Hydrographic Offices (in S-57 format). User documentation is provided in `doc/` directory.

4.3 Initialization Issues

Complete fulfillment of initialization phase is essential for assuring correct work of other procedures. All Mapper functions except `MapVersion` check, whether start-up code is successfully completed and issue an error otherwise. One function, `MapVersion` does not include this check because it is supposedly used before initialization to check the version of Mapper[2].

Calling `MapInit` is therefore required prior to any other function calls from Mapper library. `MapInit` carries out several activities that initialize internal data structures. It also registers special function to be called at program's termination via standard `atexit` *C* library function.

At first the configuration file[3] is loaded from disk and parsed. This file contains information about feature object classes (names and descriptions), colors for all color coding schemes and default values for several other parameters. User should provide syntactically correct initialization file or Mapper is unable to continue.

Initialization several data structures for presentation is the one of main tasks for `MapInit`. This includes, but is not limited to, allocating several pixmaps from X server, getting font handles from font server and reserving memory for drawing buffers. Several resources from X11 server (color cells, pixmaps and others) are to be allocated once for every display, hence `MapInit` needs to know about every display (connection to the X server), that will be used with Mapper.

Loading the Presentation Library is the last and most complex activity completed in start-up phase. Presentation Library resides in several files in directory `$MAPPER/plib/`. There are several files, defining shapes for complex line segments, outlines for cartographic symbols, patterns used to fill specific areas, and look-up tables for connecting situation model (feature object) with presentation model (its visual form). Data from those files is fetched into memory and referenced during the draw cycle.

---

[1] `libmapper.a` is actually only symbolic link to current version, `libmapper.a.2.0`. That way application need not to change its `Makefile` every time Mapper version changes.

[2] Such controls are necessary, because the syntax has changed a bit between versions 1.2 and 2.0, and probably will also have slight modifications in future releases.

[3] Sample configuration file is provided in appendix B.

4.4 File Input and Output

Modern maps contain a lot of information and occupy many megabytes on storage devices, therefore needing efficient file loading and saving routines, or the end user will be bored otherwise. As noted earlier, Mapper uses two-stage file reading (and writing) process — first original charts from Hydrographic Offices (ENC) are converted to internal representation (SENC) via separate conversion program and on the second stage these images in native format are read into memory on user's request.

Among other things, such scheme simplifies considerably the process of creating input/output filters. When adding some other file format, one should only write converter program that transforms data to Mapper format (and back), but the other routines (e.g. `MapLoad`) remain the same. At present times there are implemented two programs that transform data from DX-90 and S-57 version 3.0 to Mapper, respectively.

ENC files are encoded according to ISO 8211 standard and have to be decoded prior to any use. This process is not straightforward and as there appeared a program for exactly that purpose in the Internet, it was used instead of implementing all routines in house. The *FIPS PUB 123 Function Library* by USGS ([22]) is a free toolkit for processing files, structured by FIPS PUB 123 standard, an U.S. standard that was used as a base for ISO 8211.

The structure used for Mapper's own file format is much simpler than S-57 and does not pretend to be so universal. It is optimized to speed up loading and saving instead, having almost the same format as structures that hold the same data in memory. A call to `MapLoad` reads sequentially from disk all spatial objects (coordinates are already converted to planar system), feature objects, object classes and attributes (the last two are held separately from feature objects). Writing a map to disk involves the same operations in reverse order.

4.5 Visualizing Map Data

Drawing chart data in fast and accurate manner is the main objective of Mapper, efficiently putting stress on visualization routines. Outer interface has only a few functions (`MapDraw`, `MapAreaDraw` and corresponding procedures for drawing single object.), directly conducting the process of the picture composition, but there are many functions aiding to reach the aim.

Drawing process, started by, for example, a call to `MapDraw`, expects several preconditions to be met. First, `MapInit` should have read Presentation Library into memory and completed other initial tasks successfully. Second constraint requires setting up appropriate `MAPDISP` structure beforehand, executing `MapDispCreate` and `MapDispSet` subroutines. `MAPDISP` is a data structure that holds all information about which part of the map data set is displayed, where and how (using which scale factor) the drawing request are sent. Two members of this structure refer to Display and Drawable (these are data types defined by X), holding all necessary information about the destination of actual drawing commands. Mapper can control several screens[4] and windows simultaneously. The displays Mapper is asked to draw need not be on the same computer that runs Mapper process, they can be anywhere around the world — using the X Window System as additional layer between software and hardware gives amazing capabilities to every application.

Visible area is determined by its center point (in map coordinates), and the width and height of the pixmap (in pixels) where the drawing requests are directed. The amount of data that fits into this window is calculated, using scale factors[5] that are also part of `MAPDISP`.

The actual process of drawing contains one loop over all feature objects. To make them all appear correctly, an array of pointers to feature objects (`PA`) is created beforehand. Objects in this array appear sorted according their *display priority* (Presentation Library (S-52) specifies ten different layers for display objects.). As every map can contain thousands of objects, drawing all of them will be clearly ineffective, resulting in several "tricks", invented to exclude unnecessary objects from the display list. First, all members of classes that are not qualified for visualization (controllable via `MapClassCntl` function) are excluded. Then the current scale is compared to required attributes `SCAMIN` and `SCAMAX` for every object, determing minimal and maximal scale the object is visible, respectively. Last check looks whether the object's bounding box and visible area do intersect and otherwise drop that object.

If current object is qualified for drawing, *symbology instructions*, appropriate for that object are executed (Symbology instruction is fetched from look-up tables using class name and attribute values as filter.). A symbology instruction, according to S-52, is minimal block for generating the picture, one or more per object. There are six standard instructions — for text labels, point objects, simple line styles, complex line styles (composed from repeating symbols), color fills for areas (either opaque or transparent) and for filling area with given pattern.

Executing these instructions is quite straightforward, following the data presented in digital form in Presentation Library. IHO has developed specific language of graphical primitives for describing the small icons that symbolize chart objects (for instance buoys, anchorage places or fill pattern for airport). Instructions of that language are interpreted in Mapper at execution time and

[4]A screen consists of display adapter and monitor, several screens can be controlled with the same input devices.

[5]Scale can be different on x and y axes.

translated into drawing requests for X11. Efficiency is increased by using special cache for storing all ready-made (by interpreter function, mentioned above) pixmaps. If the requested pixmap resides already in cache it is copied straightly to the output window, bypassing time-consuming interpretation phase.

One problem that makes drawing algorithms more complex is related to the fact that X11 uses values of type `short integer` (16 bit wide on 80x86 processors), whether coordinates used by Mapper are 32-bit (`long integer`). As variables of shorter types "wrap over" when too large values are fed into them (E.g. coordinates of big areas on very large scales.), there is urgent need for "clever" algorithm that translates `long int` coordinates into `short int` ones. Getting all points of a line or area clipped correctly (including all interior border lines) was one of the most challenging tasks in Mapper, testing my skills in both geometry and programming. In addition to solving the data type problem, coordinate clipping also speeds up the whole drawing cycle, reducing considerably the amount of points to be drawn (vertices of zigzag or polygon).

Six symbology instruction classes, mentioned earlier, cannot always provide required symbolizations for all object classes. In some cases the static symbolization, taken from Presentation Library by these instructions, is not flexible enough. Visual outlook of a few feature object classes like `DEPARE` (depth areas), `LIGHTS` (various lights), and others, depend on additional factors that are not included in look-up tables in Presentation Library — safety contours, given by mariner, relations to different objects and other similar conditions. For handling such classes IHO has prepared algorithms, invoked by special symbology instruction, named *symbology procedure call.* Such procedures perform sometimes complex queries on adjacent objects or choose symbol to be displayed using user-configured parameters.

Comparing Mapper to other products, we can proudly state that by speed of drawing, Mapper outperforms most other map viewers.

4.6 Editing Capabilities

Mapper provides also basic functions for editing digital chart data, allowing to create full-featured map editors for different uses. Nevertheless, building a good editor requires more work than is needed for simple viewer — functionality provided by Mapper is quite terse.

Editing a map, either point locations or feature object attributes, is coupled with user interaction. Functions, for example, returning attributes of an object, need not be as well optimized as drawing routines, for the speed of whole process is dictated by user (as the slowest part in a chain).

Mapper contains several aids for building map editors, most of them consist of a pair of functions that allow user to ask for and set specific property of separate object, like `MapObjGetClass` and `MapObjSetClass`. The base for all following procedures is `MapObjCreate`, convenience function for making a new instance of some object class and adding it to map database. This subroutine sets all properties of newly made object, like attributes and points (If these points do not refer to any existing spatial objects, new spatial objects are also created.). To locate some already existing object, `MapGetFirstObject` and `MapGetNextObject` come handy, or one may want to use `MapGetObjByCoord` if coordinates of some point are known (E.g. when user clicks mouse button on map.). These subroutines return pointer to `FOBJ` structure[6] (feature object), mandatory for all following operations on this object.

Several functions provide a lot of data to user (and others expect it back), entitling the approach taken — make these data structures (`ATTRIBUTE`, `POINT`, `SOUNDING`, `SEGMENT` and `CONTOUR`) public and thereby result in simpler function declarations. Application programmer have to accommodate his application with these structures.

Realization of most editing functions is quite straightforward, as they provide only comfortable interface for copying data from (and to) inner structures to "user space". `MapObjGetPoints` (also `MapObjSetPoints` and `MapObjCreate`) are different, because spatial data have to be collected from spatial objects and put into an array of type `CONTOUR`. Yet public data structures are designed to contain all information that is stored in spatial objects, allowing the modified segments to be merged back into chart without losing data consistency.

`MapObjHighlight` provides an interesting tool for setting some object(s) visually apart from the rest. When `MapObjHighlight` is called with parameter *Highlight* set to 1, every geographical point, belonging to that object, is marked with orange cross inside a ring. This procedure gives comfortable method for marking user selections on the map, all the programmer needs to do is to call this subroutine and Mapper takes care that highlighting is always shown correctly by modifying the symbology instruction for given object (A non-standard symbology instruction type is invented for that purpose.).

4.7 Error Handling

Every self-esteemed software system should have some means for tracking the data flow and reporting (and/or repairing) abnormal conditions. Mapper follows this tradition and has sufficient error handling capabilities, yet they are slightly different from approaches made in other programs. As Mapper is not a complete program but only handful of subroutines, which are not

[6]Due to encapsulation, user sees it as a pointer to `void`.

meant for direct interaction with user, every application programmer who uses these procedures has to design his own methods for reporting errors and getting response to them. Mapper is connected to the rest of program via function calls and this is also the only way for reporting success and failure of these routines. Every public function has meaningful return codes and nearly all procedures can signal to their caller about abnormal situations via these codes.

The return values which signal erratic situation vary from function to function, but generally it is either -1 or `NULL` (depending on the type of return value), which is very common practice in C programming language. These values indicate, that something unexpected have happened, but this is usually not enough for localizing the cause of an error (For example, user should be given another chance for typing a file name if he does not enter the name of existing file at first.). Mapper goes one step further and provides application programmer with global variable `MapError`, holding any value from list of documented error codes (`ERRCODE` in `mapper.h`), and corresponding function (`MapErrorString`) for retrieving short message about what happened. This approach, borrowed from Unix standard functions library *libc*, gives freedom to application programmer whether his program processes this state without disturbing user (based on info in `MapError`) or displays error dialog using provided description of situation and waits for user response.

`MapError` variable is not cleared prior to successful return from any of Mapper's functions, erroneous situations should be detected by return values. Such scheme works well for all procedures but a few. Namely for object searching functions (`MapGetFirstObject` and friends), return value of `NULL` could indicate either an error or simply the fact that there were no more objects on map. Therefore programmer should set the value of `MapError` to 0 before calling such procedures and examine it afterwards, when value other than 0 indicates abnormal termination.

Every subroutine in Mapper checks the validity of its input data (whether all arguments have values in meaningful range) and also several preconditions (initialization of Mapper, completition of `MapGetFirstObject` before calling `MapGetNextObject`, etc.). Several procedures use also silent fallbacks if some condition is not met and give up with an error only if all else fails (E.g. if the visual that user requested is not available, similar types are evaluated.).

It should be noted that there are conditions where input parameters are invalid but Mapper cannot detect this (like pointers to already freed memory block). These situations may cause unstable behavior and program termination by operating system or windowing system. The application programmer should be aware of, and put additional checks on every function's parameters to avoid such situations.

4.8 Testing and Quality Control

There are several methods used during the development process to ensure maximal correctness and performance of software. These techniques are either subjective (carried out by humans) or objective. Objective quality assurance methods include careful observation of compiler warnings and continuous use of dynamic memory debugger at development time. The `MEM` package, written by W. Bright and released to public domain, is a set of functions used for debugging C pointers and memory allocation problems. It sets up additional checks to standard dynamic memory handling functions, reducing by large the occurrence of errors with pointers ("pointer bugs").

The list of subjective quality management methods include frequent self-control of programmers (usually in `mtest` testbench), eliminating most known bugs and making sure that software corresponds to standards. There are also regular meetings inside the development team (2–3 persons), when the work of programmers is reviewed and criticized if necessary. The small number of persons involved in this project has proven optimal in cooperation and "in depth" understanding of important standards and Mapper software.

Performance and functional testing was usually carried out in `mtest` environment but from time to time `PPHMS` system was used for field tests (Requirements for speed and efficiency were evaluated on board of measure ship in real situations.). The parts of Mapper which were done under contract with Estonian National Maritime Board were additionally reviewed and tested by representatives of ENMB.

Chapter 5

Using Mapper in Applications

5.1 PPHMS — Multi-channel and Multi-echo Hydrographic Survey System Software

As stated in chapter 3, the Mapper project came into existence when there raised a need for visualizing nautical chart data for PPHMS. The development and continuous enhancement of PPHMS has had a great influence to Mapper, being the major force driving Mapper to the position where it is today.

The PPHMS system[18, 17] consists of several separately manageable windows ("views"), each having different data displaying capabilities. Some of them — data visualizer AV (Area View, figure 5.1 on the following page.), Plan Editor PE, Object Editor OE (Figure 5.2 on page 54.) and NaviView NV — need to present cartographic background information. The maps, drawn by Mapper are considered by PPHMS as lying at the very bottom layer of display, onto them are drawn rectangular grid with distance marks (easting and northing) and on top view-specific data — ship's current and past positions, actual depth data from sonars, survey plan, etc.

According to the requirements of Estonian National Maritime Board, survey data and maps can be visualized either in Mercator or Transverse Mercator projections. As the survey system receives current vessel's coordinates by differential GPS receiver, the WGS 84 (a datum used in GPS) was a natural choice for a geodetic datum. The necessary parameters (projection, central meridian, coordinate system starting point etc.) are written to PPHMS configuration files and fed to appropriate geodetic transformation procedures during initialization. All positioning and map data is treated in cartographic (planar) coordinates inside PPHMS, but written to output files in geodetic coordinates (latitudes and longitudes), to make data easily available to other systems.

There can be two maps loaded at the same time in PPHMS, one covering another. First of them, the *background map* is static and cannot be modified, usually existing nautical map is used for that purpose. The other is *map of*

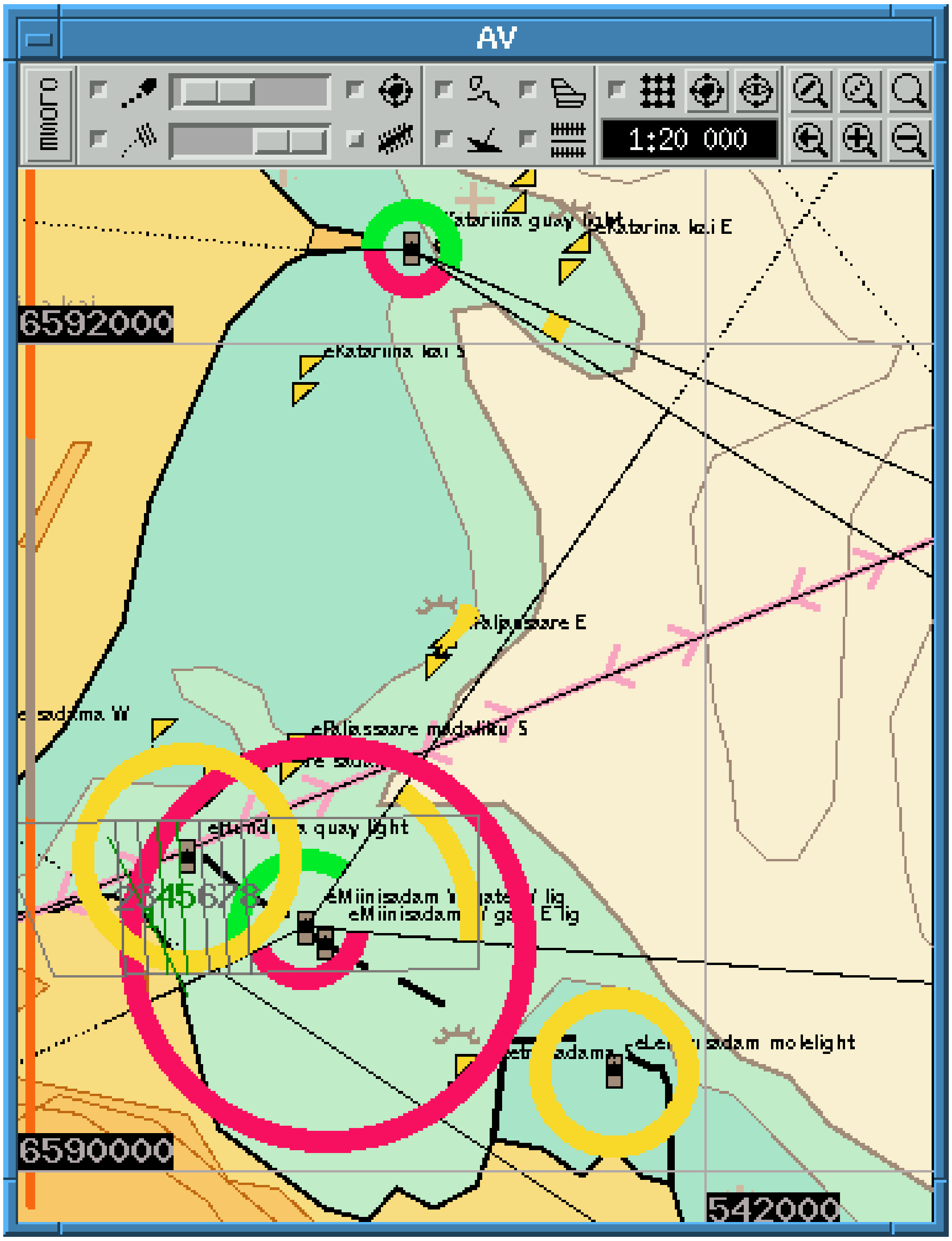

Figure 5.1: Area View AV window of PPHMS. Hundipea harbor (Tallinn). Simplified symbol set.

modifications or *objects map*, laid on top of the background map. This chart is usually composed of point objects only, which are added or modified on board of the vessel. For comfortable use these two maps should cover the same area and have the same scale. The user can toggle visible cartographic classes separately for foreground and background maps. Other operations — zooming and centering — affect both maps simultaneously. Surveyor can zoom chart (and survey data displayed on top of maps) in several ways. First, he can select item from a set of predefined values, effectively setting display scale to that value. Alternatively, he can press down and drag a mouse button, zooming the selected area to window size. Changing a center point of display is also done this way, by clicking to the point to be centered. Hydrographer can also zoom visible area in or out by pressing appropriate pushbuttons. There are also functions for measuring distance and angle between any two points on map or between current vessel location and some other point.

The maps (background and objects map) are read into system via configuration panel and are common to all windows, however, each view has its own set of controls for zooming and centering. The system-wide configuration panel also includes controls to adjust presentation of maps (selection of visible object classes, changing own ship safety depth, color scheme and other parameters).

The PPHMS system also includes a separate screen of NV for helmsman. This view is supposed to be managed onto separate monitor on the bridge to assist steering of ship. Helmsman only receives information (ship position, course, speed, etc.) from that display, all necessary manual adjustment (e.g. setting correct map scale) is done by surveyor — actions he takes reflect both on hydrographer's and helmsman's displays. To minimize expenditures the helmsman's monitor is connected to second display adapter in hydrographer's computer, both screens are managed by one multi-headed X server.

5.1.1 Object Editor OE

Object Editor OE (Figure 5.2 on the following page.) is a tool for hydrographer for adding or modifying objects on a map. These modifications (Getting more accurate position of a wreck, measured with GPS, or noticing a suspicious obstruction, for example.) are done on board of survey ship, to make charts corresponding to current situation. Later these adjustments are collected together and handed to cartographers who put them into official database and then to printed and electronic maps.

OE is more integrated with Mapper than other parts of PPHMS, being the only view that uses map editing capabilities. All changes are done on the objects map and reflect instantly on all other views that have map of modifications switched on.

Object Editor handles only point objects (An object whose location is

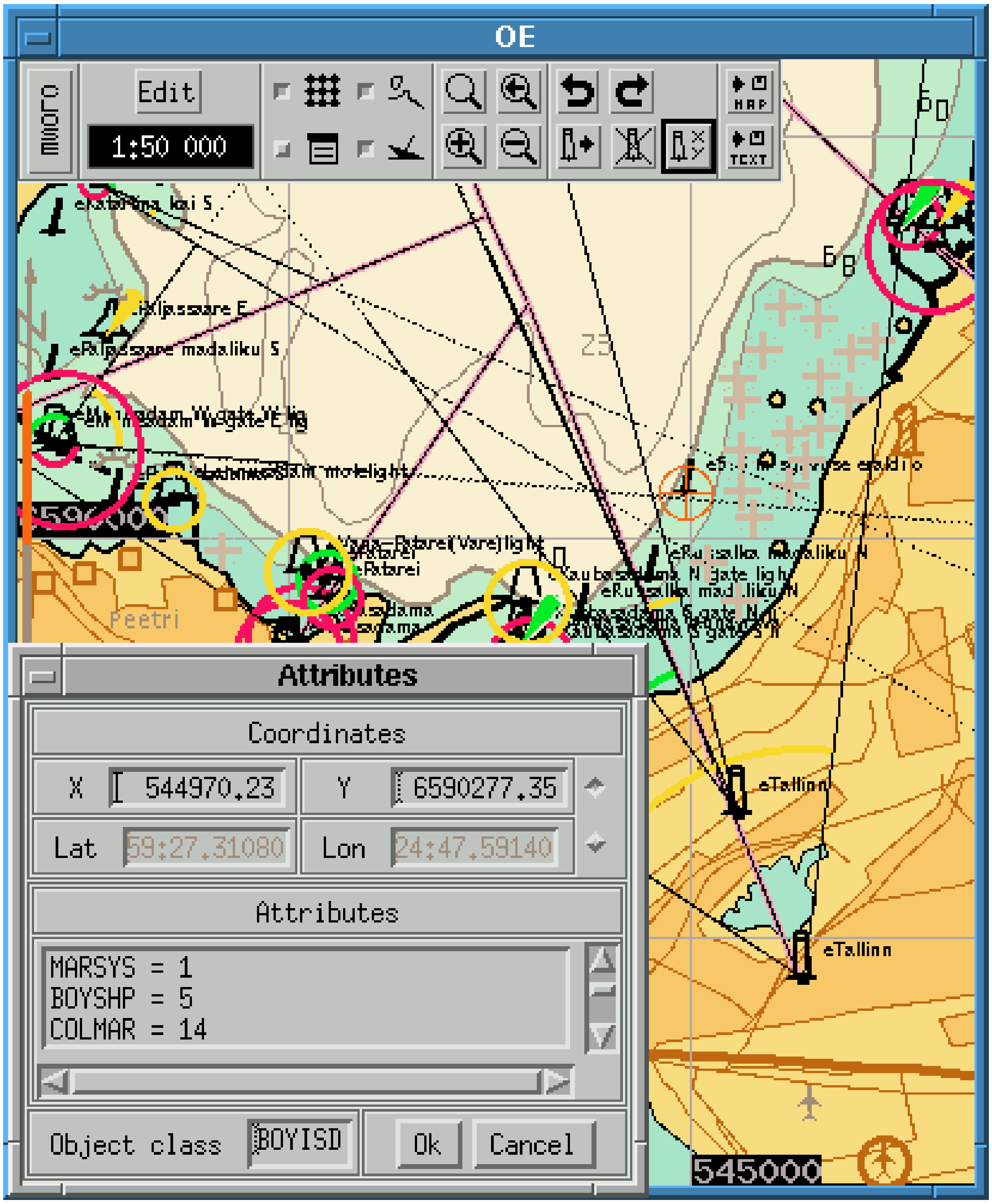

Figure 5.2: OE screenshot. Traditional (INT 1) symbol set. Tallinn (Estonian Nautical Chart sheet no. 507).

given with only one coordinate pair.), such as buoys, beacons, rocks, wrecks and obstructions. User can add a new object to map, delete or move an existing one or change its attributes, class or coordinates via dialog box. Every point object on map can be selected for these operations in two ways. First method is clicking mouse button on its location. Sometimes it is not easy to locate exact reference point from the shape of object's presentation. As an alternative, one can also pick an item from a list, sorted by classes. The selected object is distinguished from others by orange ring across the object's location, and can be automatically centered. As specified in S-57, attribute `SORDAT` is added (or updated) every time some property of an object is changed, which makes easier tracking of changes. `OE` features also a ten-level undo/redo system for object manipulations (add/remove/change). The output is written to map files in Mapper format or in human-readable text files.

5.2 Mapper Testbed `mtest`

`mtest` (see figure 5.3 on the next page) is a simple program whose main purpose is to serve as a testbench during the development of Mapper library. It is graphical front end for Mapper functions, features are added in conjunction with enhancement of Mapper. Currently user have influence to most of map viewing functions but it is expected to evolve into complete map editor in the future. As there is no commercial interest in this project (at least at the moment), any particular function is added to `mtest` only when it is implemented (or refined) in Mapper and there raises a need for testing.

Currently `mtest` can read digital maps in Mapper's native format and display them using specified scale and center point. There are menu choices for zooming in and out (a zoom-in function can be carried out also by mouse, as in `PPHMS`) and fitting the whole map to visible window. Classes of objects can be selected for display from list. There is also a dialog for changing several presentation parameters and color reference box.

User can also highlight and query any objects on map. For selected object all relevant information is displayed (Geometric primitive — point, line or area, coordinate, attribute names and values that are attached to that object.).

5.3 Radar Indicator Panel `ASD`

Another product from R-Süsteemid that uses Mapper is a tool for airport dispatchers, helping to locate current positions of air vehicles, called `ASD` — Air Situation Display (See figure 5.4 on page 57.). `ASD` features include display of targets from multiple radars either as mosaiced (each radar separately) or multi-radar image (targets from different radars are matched), support for

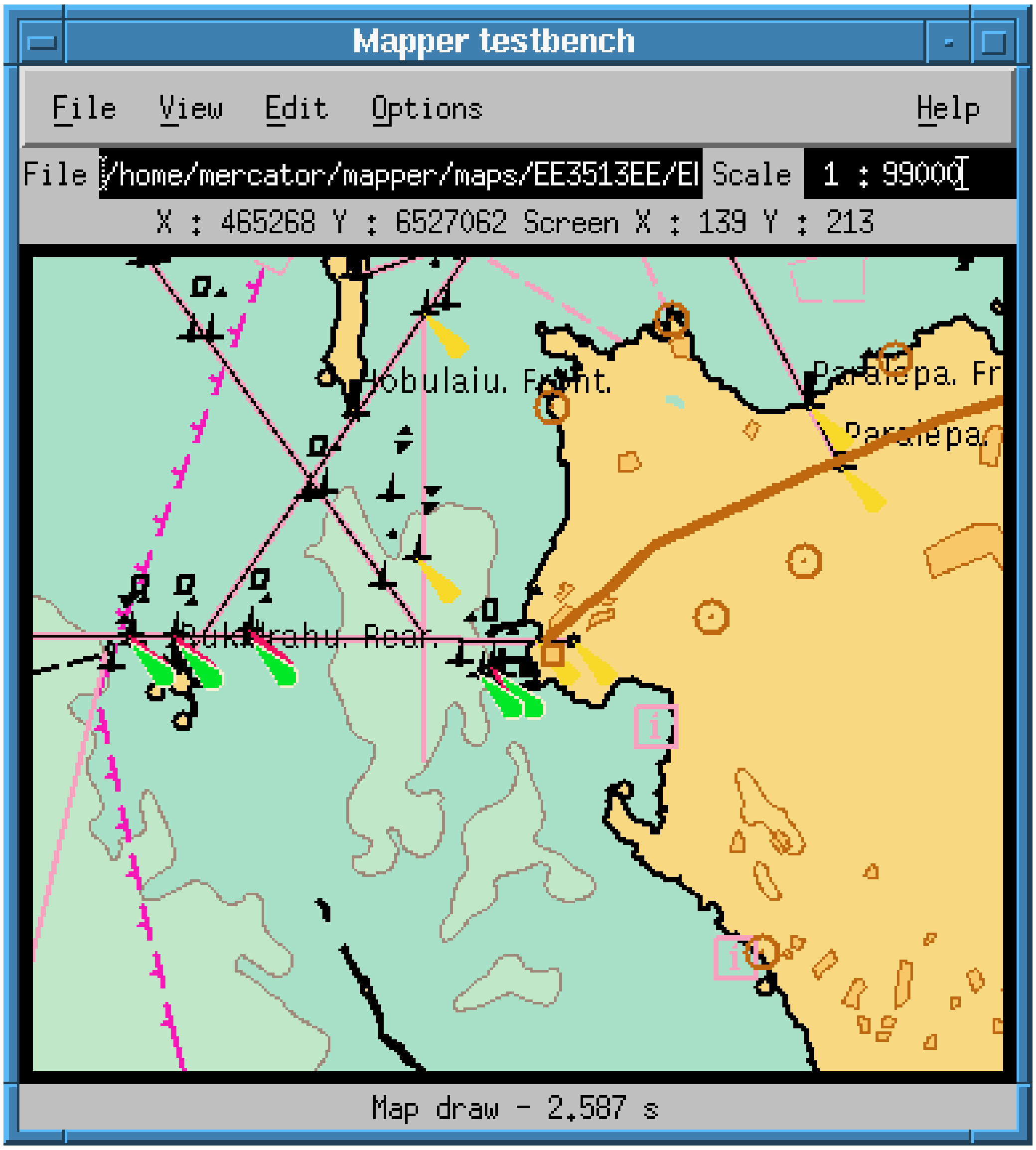

Figure 5.3: Screenshot of `mtest`, a map viewer. Rohuküla harbor.

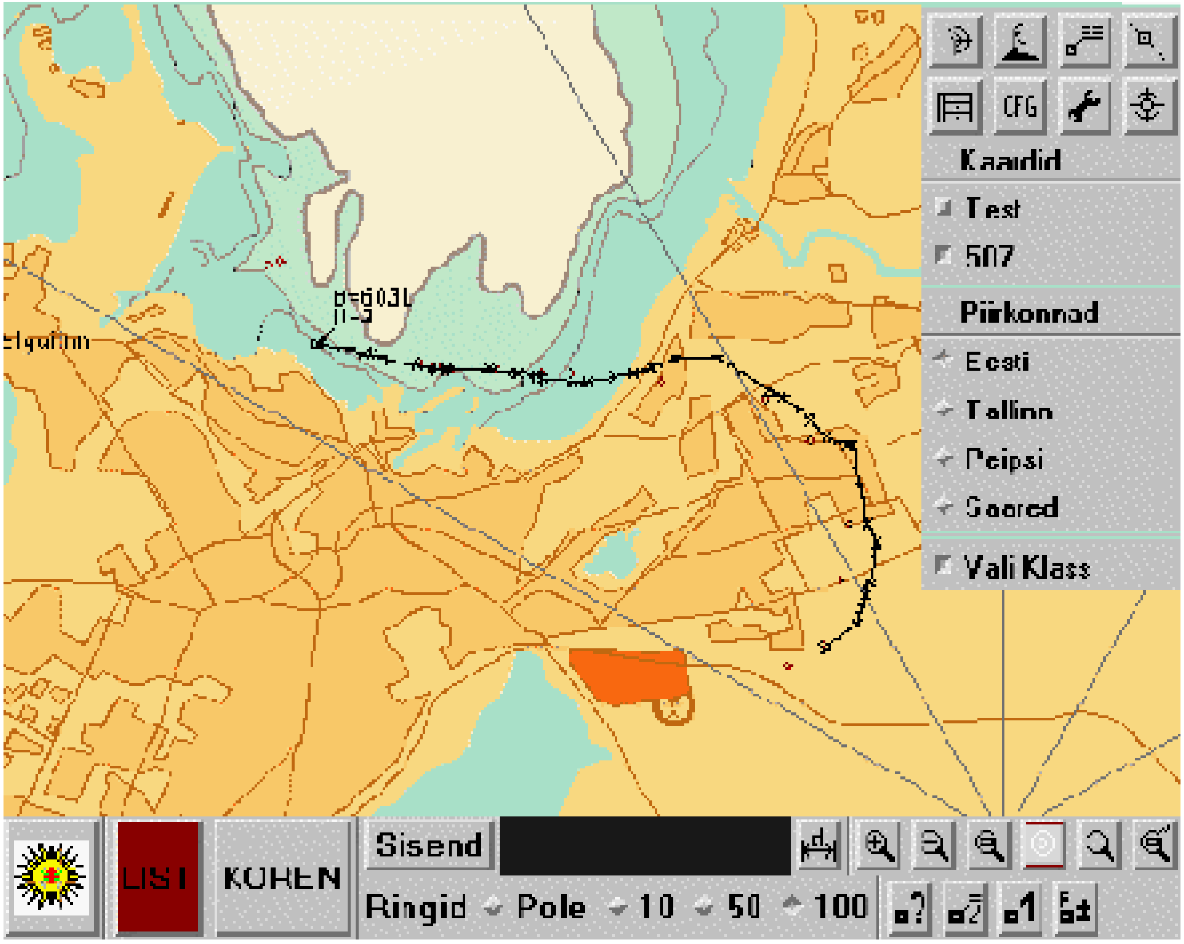

Figure 5.4: Air Situation Display, Tallinn airport.

primary and secondary radars, and correct geographical reference of targets from all radar plots. Care is taken to eliminate undesirable artifacts that are often included in raw radar data (e.g. reflections, missing data, etc.).

ASD uses complex mathematical apparatus, from statistics to automatic control theory, for achieving maximal accuracy on tracking the positions of airplanes. System uses adaptive Kalman filters, first to predict track of every target and then again applies the same filter to the output of *virtual radar* (Combined data from two or more radar devices with identical targets linked together.). Different algorithms are used, after predicting tracks with above method, for ascertaining and solving special conditions, such as manoeuvres, for which ordinary Kalman filter gives poor and unpredictable results.

To help airport dispatchers locate aircraft positions more easily, ASD has also support for background maps. However, as Mapper currently supports only charts in S-57 format, the availability of appropriate data puts severe constraints on this feature. With ASD system user can load different maps and select classes he wants to be visible. Zooming and panning are also supported.

ASD is installed as a part of more complex software and hardware system in Dzerzhinsk, Nojabrsk, Pulkovo and Samara radar stations.

Chapter 6

Conclusion

The Mapper project is an ongoing effort with quite long "to do" list. Surely the most valuable resource is time — although the clock frequencies of computers are constantly raising, software engineers cannot have just vacation, because customers and managers always require performance that is accessible only by tweaking and optimizing source code.

Some problems remain in current version, most of which will probably be fixed after completing the version 2.0 of Mapper. As noted earlier, the compliance to standards is not at the level it should be. There are also some (quite technical) problems with X11, putting restrictions to visualization or slowing down the drawing process.

The list of possible addenums (or wishes) contains, in no particular order, following :

- Porting Mapper to other platforms. Microsoft Windows is an attractive niche, due to its large market share. Producing Windows version of Mapper would not be a trivial task, as the logic of Microsoft Windows differs significantly from that of X Window System, probably requiring considerable redesign and full rewrite of code in C++ or some other language that has support for Microsoft Windows Application Program Interface.

- Display PostScript. This system, which is quite common supplement on top of X11 (or possibly other windowing systems), offering much more oportunities (using the full apparatus of PostScript language) to control display output than plain X.

- Ability to process files in different formats. Flexibility gained from data intercahnge with other GISes (like ARC/INFO and Microstation) will help to use Mapper in even wider range of applications.

- Multi-threading. The ability to use lightweight threads will solve some problems that remain in current implementation, where all Mapper code runs as single process. For example, an application using several charts gains better performance if it can differentiate handling of different maps into separate threads. Mapper is not thread-safe now, reaching this would take fair amount of work (eliminating global variables etc.), but the result is probably worth it.

- Better conformance to IHO standards. Currently there are some less important areas of both S-57 and S-52 where Mapper's behaviour does not follow standards very strictly. On the other hand, these improvements would considerably slow down processing of maps. A non-exhaustive list of missing features includes correct visualization of all S-57 object classes, color calibration according to monitor, ability to load and save multi-cell charts and checks for permitted class/attribute combinations.

The story of Mapper has been quite successful so far and has given lots of new skills to the developers. But everyone hopes the future will be even brighter.

Glossary

Area object Cartographic object, composed from a set of points, treated as vertices of a polygon.

ASD **A**ir **S**ituation **D**isplay. Software system from R-Süsteemid for processing multiple radar data. Locations of air vehicles are displayed graphically on top of a map data, using adaptive multiradar tracking.

Attribute A property of feature object with name (common to all instances of a class) and value (specific to given object).

Base Map See Estonian Base Map.

Basic Map See Estonian Basic Map.

Cell Piece of logically connected chart data. Any map contains one or more cells.

Chart See map.

Display Logical view of a computer workstation in X Window System, consisting of one set of input devices (keyboard, mouse, etc.) and one or more monitors (Screens).

DX-90 Data exchange format for S-57 edition 2.0. At present the up-to-date version is S-57, edition 3.0 and the exchange format which goes with it is also named S-57.

Easting Horizontal distance on a grid system from coordinate origin. An x coordinate of a point on planar map.

ECDIS **E**lectronic **C**hart **D**isplay and **I**nformation **S**ystem. A GIS for sailors for vessel navigation (instead of paper charts). Must complain several standards by IHO and IMO.

ECS **E**lectronic **C**hart **S**ystem. A system for displaying electronic maps. Simpler than ECDIS and cannot be used as the only source for navigation information.

Ellipsoid Mathematical figure, closely approaching the geoid in form and size, used as a surface of reference for geodetic surveys. A reference spheroid or ellipsoid is a spheroid, determined by revolving an ellipse about its shorter (polar) axis and used as a base for geodetic surveys of a large section of the Earth.

ENC **E**lectronic **N**autical **C**hart. Official digital map, following the IHO Special Publication S-57 and issued by Hydrographic Office.

ENMB **E**stonian **N**ational **M**aritime **B**oard (*Eesti Veeteede Amet — EVA* in estonian). Estonian state enterprise responsible for safety of navigation in Estonian waters.

Equator Parallel with zero latitude, a circle on the surface of the Earth. The Equator lies on the plane, perpendicular to the Earth's spin axis and crossing it in midway from one pole to another.

Estonian Base Map A map with scale 1:50 000, produced by Estonian Map Centre from satellite images.

Estonian Basic Map Map for the whole area of Estonia, produced by Estonian Map Centre. Digital map has scale (resolution) 1:10 000, printed sheets are scaled to 1:20 000.

Feature object Structural element of chart, having descriptional information (class name, attributes) and references to spatial objects (location).

FIPS **F**ederal **I**nformation **P**rocessing **S**tandard. An official standard of U.S.A. in fields of information interchange and processing.

Geodetic datum Traditional term for a reference system.

Geoid Figure of the Earth, visualized as a mean sea level surface extended continuously through the continents. It is a theoretically continuous surface that is perpendicular at every point to the direction of gravity (the plumbline).

Geometric primitive Property of feature object, reflecting its spatial extent and shape. Either point, line or area.

GIS **G**eographic **I**nformation **S**ystem. A computer controlled database and administration system, consisting of hardware, software, data and applications. With this system it is possible to record, save, manage, model and analyze spatial data and present them in an alphanumeric and graphic way.

GPS **G**lobal **P**ositioning **S**ystem. A satellite based positioning system.

Great circle A circle on a sphere, such that the plane containing the circle passes through the center of the sphere. A segment of a great circle is the shortest way between any two points on the sphere.

Grid Network of uniformly spaced parallel lines intersecting at right angles. When superimposed on a map, it usually carries the name of the projection used for the map — Lambert grid, Transverse Mercator grid, Universal Transverse Mercator grid.

IERS **I**nternational **E**arth **R**otation **S**ervice. An organization coordinating the values of reference constants for the Earth.

IHO **I**nternational **H**ydrographic **O**rganization is an intergovernmental consultative and technical organization working to support the safety of navigation and the protection of the marine environment.

INT 1 Symbols, abbreviation and terms used on paper nautical charts, specified by IHO.

IMO **I**nternational **M**aritime **O**rganization is the United Nations' specialized agency responsible for improving maritime safety and preventing pollution from ships.

IRM **I**ERS **R**eference **M**eridian. The reference (zero) meridian, defined by IERS.

IRP **I**ERS **R**eference **P**ole. A reference point, defined by IERS as a point of intersection of the mean spin axis and the surface of the Earth.

ISO 8211 Standard from **I**nternational **S**tandardization **O**rganization for system-independent file format for data transformation.

ITRF **I**ERS **T**errestial **R**eference **F**rame. A global geodetic datum, maintained by IERS, having an accuracy below 10 cm.

Latitude Angular distance, in degrees, minutes, and seconds, of a point north or south of the Equator.

Line object A cartographic object which is composed from straight segments between its points (an open or closed zigzag).

Longitude Angular distance, in degrees, minutes, and seconds, of a point east or west of the zero (Greenwich) meridian.

Map A collection of feature and spatial objects, connected to cells according to specific rules. Terms chart and map are treated as synonyms in current work.

Map projection A systematic representation of all or part of the surface of a round body, especially the Earth, on a plane.

Mercator A cartographer of 16^{th} century, also the widely used map projection he invented.

Meridian Great circle on the surface of the Earth passing through the geographical poles and any given point on the Earth's surface. All points on a given meridian have the same longitude.

Northing Vertical distance on a grid system from coordinate origin. A y coordinate of a point.

Parallel A circle, or approximation of a circle, on the surface of the Earth, parallel to the Equator and connecting points of equal latitude.

Pixmap X11 data type, presented to programmer as a canvas for drawing requests.

Point object A cartographic object whose location is given with only one pair of coordinates.

Polar motion The motion of the rotation axis of the Earth, causing the location of geodetic pole to move around.

PPHMS Multi-channel and Multi-echo Hydrographic Survey System Software (***P****aljukajalise ning* ***P****aljukanalilise sonariga* ***H****üdrograafiliste* ***M****õõtmiste teostamise juhtimise ja jälgimise* ***S****üsteem* in Estonian). An online system for carrying out hydrographic surveys, developed by R-Süsteemid Ltd.

Presentation Library A collection of machine readable files and accompanying documentation that links every object class and attribute of the SENC to the appropriate presentation.

Presentation model Guidelines for displaying map data.

Quadrangle Four-sided area, bounded by parallels of latitude and meridians of longitude, used as an area unit in mapping (dimensions are not necessarily the same in both directions).

Reference frame A set of control points to realize a reference system.

Reference surface Mathematically (e.g. sphere or spheroid) or physically (geoid) defined surface to approximate the shape of the Earth for referencing the horizontal and/or vertical position.

Reference system Shape, size, position and orientation of a (mathematical) reference surface, (e.g. sphere or spheroid). Normally it is defined in a superior, geocentric three-dimensional coordinate system (for example WGS 84).

S-52 IHO Special Publication no. 52. Specification for Chart Content and Display Aspects for ECDIS.

S-57 IHO Special Publication no. 57. IHO Transfer Standard for Digital Hydrographic Data (object catalogue, data model, transfer format).

Screen Structure (X11), describing logical monitor (and display adapter board), connected to a workstation.

SENC **S**ystems **E**lectronic **N**autical **C**hart. Data in ECDIS private format, generated from ENC.

Situation model Data structures and related activities to handle spatially referenced data efficiently.

SOLAS The International Convention for the **S**afety **O**f **L**ife **A**t **S**ea. The convention in force, known as SOLAS 1974, covers various aspects of ship safety, including construction, fire protection, life-saving appliances, radio communications, safety of navigation, the carriage of cargoes and safety measures for high speed craft.

Spatial object Container for set of related points in map.

Spheroid See ellipsoid.

SQL **S**tructured **Q**uery **L**anguage. Standardized language for database queries.

TM **T**ransverse **M**ercator. A map projection invented by J. H. Lambert.

USGS **U**nited **S**tates **G**eological **S**urvey. An organization providing reliable, impartial information to describe and understand the Earth.

UTM **U**niversal **T**ransverse **M**ercator. A map projection (very similar to TM) and grid system covering the whole world.

WGS **W**orld **G**eodetic **S**ystem. A geodetic system, providing the basic reference frame and geometric figure for the Earth, also models the Earth gravimetrically, and provides the means for relating positions on various local geodetic systems to an Earth-centered, Earth-fixed (ECEF) coordinate system.

X Window System, Version 11 A network transparent window system running on a wide range of computing and graphics machines. Also known as X11.

Index

Bibliography

[1] Activities. http://www.iho.shom.fr/activ.html.

[2] Alexander, L., and Spalding, J. W. Integrated marine navigation systems of the future. Presented at the Institute of Navigation National Technical Meeting, San Francisco, CA, January 20, 1993.

[3] Dana, P. H. Geodetic datum overview. http://www.utexas.edu/depts/grg/gcraft/notes/datum/datum.html, 1997.

[4] IMO and the safety of navigation. http://www.imo.org/focus/safnav/safstart.htm.

[5] International Hydrograhics Organization. *Colour & Symbol Specifications for ECDIS*, 4th ed. International Hydrographic Bureau, Monaco, 1997.

[6] International Hydrographic Organization. *IHO Transfer Standard for Digital Hydrographic Data. Edition 3.0.* International Hydrographic Bureau, Monaco, Nov. 1996.

[7] ivtools home page. http://www.vectaport.com/ivtools/index.html, 1997.

[8] Jürgenson, H. Eesti kaardisüsteemi matemaatiline alus. *Geodeet 8(32)* (1995), 3–5.

[9] Mark, D. M., Chrisman, N., Frank, A. U., et al. The GIS history project. http://www.geog.buffalo.edu/ncgia/gishist/bar_harbor.html.

[10] Mayr & Kockelhoff, Inc. Mayko Xmap. http://www.mayko.com/xmap/.

[11] Moore, T. Coordinate systems, frames and datums. http://www.arkgeo.com/paper.html.

[12] Moore, T. Where are you now? Are you sure? http://www.arkgeo.com/slides.html, 1997.

[13] A/O Моринтех. *dKart base line.*

[14] Oidjärv, H. Eesti Vabariigi uued merekaardid. *Geodeet 8(32)* (1995), 35–38.

[15] The Perry-Castañeda library map collection. `http://www.lib.utexas.edu/libs/PCL/Map_collection/Map_collection.html`, 1998.

[16] Potter, H. Eesti uus põhikaart. *Geodeet 2(26)* (1992), 6–8.

[17] OÜ R-Süsteemid. *PPHMS. Multi-channel and Multi-echo Hydrographic Survey System*, 1997.

[18] Rüütel, M. Meremõõdistussüsteemi tarkvara loomine. Diplomitöö, Tallinna Tehnikaülikool, juuni 1997.

[19] Scheuermann, W. What is ECDIS and what can it do? `http://www.sevencs.com/7cs/aboutECDIS/ecdis_en.html`, 1997.

[20] SevenCs GmbH. EC2007. ECDIS kernel product information. `http://www.sevencs.com/7cs/products/eckernel.html`, 1996.

[21] Snyder, J. P. *Map Projections — A Working Manual.* U. S. Geological Survey Professional Paper 1395. United States Government Printing Office, Washington, DC, U.S.A., 1987.

[22] U. S. Geological Survey. *Programmers Reference Manual for FIPS PUB 123 Function Library*, Mar. 1993.

[23] The Universal Transverse Mercator (UTM) grid. Fact sheet 142–97. `http://mapping.usgs.gov/mac/isb/pubs/factsheets/fs14297.html`, 1997.

[24] Viru, R. Baaskaardi olemusest. *Geodeet 8(32)* (1995), 30–34.

[25] Voser, S. A. Important to know about map projections and geodetic datums. `http://habicht.bauv.unibw-muenchen.de/voser/bezug/prj_vs_datum.html`, 1996.

[26] World Geodetic System 1984 (WGS 84). `http://www.afmc.wpafb.af.mil/organizations/HQ-AFMC/IN/mist/wgs1984.html`, 1995.

[27] Wilford, J. N. Revolutions in mapping. *National Geographic 193*, 2 (Feb. 1998), 6–39.

Appendix A

Definition of the Public Interface — `mapper.h` header file

The `mapper.h` header file contains definition of the whole public interface of Mapper — constants, definitions of data structures and variables and function prototypes.

```
/************************************************************
mapper.h

eli.wings.ee
Tue Feb  3 12:07:28 EET 1998
Thu Mar  5 16:12:41 EET 1998
Wed Apr 22 13:20:11 EET DST 1998

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

#ifndef MAPPER_H
#define MAPPER_H 1

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

#ifndef _XLIB_H_
#include <X11/Xlib.h>
#endif

#ifndef _INTRINSIC_H_
#include <X11/Intrinsic.h>
#endif
```

```
/*********************************************************/
/*
    max nr of different displays or screens available
*/
#define MAX_DISPLAYS 4

/*
    actual attribute and class name lengths
    MUST BE _smaller_ BY ONE (i.e. 6 characters in both cases)
*/
#define ATTR_NAME_LEN 7
#define CLASS_NAME_LEN 7
#define COLOR_NAME_LEN 6

/*
    situation model codes
*/
#define SM_UNKNOWN 0
#define SM_S57 1

/*
    presentation model codes
*/
#define PM_UNKNOWN 0
#define PM_S52 1

/*
    flags to be ORed together for MapSetPresentationParams
    or MapGetPresentationParams
*/
#define PP_NORTH_ARROW (1L<<0)
#define PP_TWO_SHADES (1L<<1)
#define PP_SHALLOW_PATTERN (1L<<2)
#define PP_TEXT_OFF (1L<<3)
#define PP_SAFETY_DEPTH (1L<<4)
#define PP_SHALLOW_CONTOUR (1L<<5)
#define PP_SAFETY_CONTOUR (1L<<6)
#define PP_DEEP_CONTOUR (1L<<7)
#define PP_SYMBOL_SET (1L<<8)
#define PP_VISUALS (1L<<9)
#define PP_COLOR_MAPS (1L<<10)

/*
```

```
    diffrent color coding schemes
*/

#define CCS_DAY_BRIGHT         0
#define CCS_DAY_BLACKBACK      1
#define CCS_DUSK_BLACKBACK     2
#define CCS_NIGHT_FILTERED     3

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

/*
    error codes from Mapper functions (in MapperError
    variable)
*/
typedef enum errcode {
    MNOERROR = 0,   /* sucess */
    MFOPEN  =  1,   /* error opening file*/
    MFREAD  =  2,   /* error reading file */
    MMAP    =  3,   /* not a valid map structure */
    MDISP   =  4,   /* not a valid mapdisp structure */
    MOBJ    =  5,   /* not a valid obj structure */
    MSINSTR =  6,   /* not a valid symbolization instruction */
    MIDISP  =  7,   /* request for too many displays */
    MINIT   =  8,   /* mapper not properly initialized */
    MNOOBJ  =  9,   /* map contains no objects */
    MFTYPE  =  10,  /* unknown feature obj type */
    MFNEXT  =  11,  /* findnext without findfirst */
    MIPRES  =  12,  /* cannot init presentation model */
    MNOMEM  =  13,  /* cannot alloc memory */
    MNOCOLOR = 14,  /* no such color */
    MCONSISTENCY = 15,  /* consistency have been lost */
    MCSCHEME = 16,  /* invalid color coding scheme */
    MNCLASS = 17,   /* unknown object class */
    MRCOLORS = 18,  /* cannot read RGBs from file */
    MRSYMBOLS = 19, /* cannot read symbols */
    MHASH = 20, /* error creating hash table */
    MIATTRN = 21,   /* invalid attribute */
    MIPRESENTATION = 22,    /* invalid presentation model */
    MISITUATION = 23    /* invalid situation model */
} ERRCODE;

typedef
struct  attribute {
    char Name[ ATTR_NAME_LEN ];
```

```
    char Type;
    union {
        char *String;
        long Number;
    } Value;
} ATTRIBUTE;

typedef struct point
{
    long x;
    long y;
} POINT;

typedef struct sounding
{
    long x;
    long y;
    long d;
} SOUNDING;

typedef struct segment
{
    long ID;    /* segment identifier !!!READ_ONLY */
    int Type;   /* type of segment 1 - 3D (sounding) 0 - 2D */
/*
    number of points in segment
*/
    unsigned long PointCount;
    union
    {
        POINT Point;
        SOUNDING Sounding;
    } *Points;
} SEGMENT;

typedef struct contour
{
/*
    0 - exterior (or line or point)
    1 - interior
*/
    int Type;
/*
    number of segments in contour
```

```
*/
    unsigned long SegmentCount;
    SEGMENT *Segmets;
} CONTOUR;

typedef struct pparams
{
/*
    toggles presentation of north arrow and scalebar
*/
    int NorthArrow;
    int TwoShades;  /* only two-color depth areas */
    int ShallowPattern; /* shallow water highlight */
    int SymbolSet;  /* 0 - simplified / 1 - traditional */
    int TextOff;    /* 1 if text is switched off */

    short SafetyDepth; /* selected safety depth [m] */
/*
    selected shallow water contour [m]
*/
    short ShallowContour;
    short SafetyContour; /* selected safety contour [m]*/
    short DeepContour; /* selected deep water contour [m]*/

/*
    visuals to be used in creating windows
    !!! Read-only
*/
    Visual  *Visuals[ MAX_DISPLAYS ];
/*
    colormaps to be used in creating windows
    !!! Read-only
*/
    Colormap ColorMaps[ MAX_DISPLAYS ];
} PPARAMS;

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

/*
    returns version of the mapper library
    mapper needs not to be initialized
    major version number is in higher word (16bit),
    minor version in lower word
*/
```

```
unsigned long MapVersion( void );

/*
    initializes mapper
    all other functions depend on successful completition of
    mapinit()
*/
int MapInit( unsigned char DspCnt, Display *display[],
    int SituationModel, int PresentationModel,
    const char *ConfigFile );

/*
    get names and descriptions of all available map object
    classes
    return total number of classes
*/
int MapGetClassNames( char ***Classes, char ***Descriptions );

/*
    returns string of last error message
*/
char *MapErrorString( void );

/* sets pixel per meter values for both axes */
int MapSetPixPerM( unsigned char DspNr,
    long PixPerMx, long PixPerMy );

/* sets or resets visibility flag on class ClassName */
int MapClassCntl( void *Map,
    const char ClassName[ CLASS_NAME_LEN ], int Visibility );

/* sets or resets visibility flag on all classes */
int MapCntl( void *Map, int Visibility );

/* allocates MapDisp structure */
void *MapDispCreate( void );

/* set map display center & zooming factor */
void *MapDispSet( void *MapDisp, int DspNr, Window Win,
    short WinCX, short WinCY, long MagF,
    long MapCX, long MapCY, double MPX, double MPY );

/* frees MapDisp structure */
int MapDispDestroy( void *MapDisp );
```

```
/* draws map to display area MapDisp */
int MapDraw( void *Map, void *MapDisp, int Changed );

/*
    draws map to display area MapDisp
    p1 (UL) & p2 (LR corner) specify the clipping rectangle
*/
int MapAreaDraw( void *Map, void *MapDisp, int Changed,
    POINT p1, POINT p2 );

/* draws one object according to MapDisp */
int MapObjDraw( void *Map, void *Obj, void *MapDisp,
    int Changed );

/*
    draws one object according to MapDisp
    in clipping rectangle (UL p1, LR p2)
*/
int MapObjAreaDraw( void *Map, void *Obj, void *MapDisp,
    int Changed, POINT p1, POINT p2 );

/*
    returns pixel value of requested map color or
    -1 on error
*/
Pixel MapGetColor( void *MapDisp,
    const char colorname[ COLOR_NAME_LEN ]);

/*
    return pixel values and color names of active color
    coding scheme.
    application is responsible for freeing memory allocated
    for colors_return and color_names_return
*/
int MapGetColors( void *MapDisp, Pixel **ColorsReturn,
    char ***ColorNamesReturn );

/*
    returns current color coding scheme or negative value on
    error
*/
int MapGetColorScheme( void *MapDisp );
```

```
/*
    sets current color coding scheme to specified value.
    use defined constants CCS* for Sceme's value
*/
int MapSetColorScheme( void *MapDisp, unsigned char Scheme );

/* set map presentation parameters */
int MapSetPresentationParams( void *MapDisp,
    unsigned long ParamMask, PPARAMS *Params );

/* get map presentation parameters */
int MapGetPresentationParams( void *MapDisp,
    unsigned long ParamMask, PPARAMS *Params );

/*
    allocate memory for map structure
    returns pointer to map structure
    have to be called prior to any operation with map
*/
void *MapCreate( void );

/* load map data from file MapName.map */
int MapLoad( void *Map, const char *MapName );

/* frees all map data for new MapLoad */
int MapUnLoad( void *Map );

/* saves map to file MapName.map */
int MapSave( void *Map, const char *MapName );

/* frees map structure */
int MapDestroy( void *Map );

/* request coordinate magnification factor from map */
long MapGetCoordinateMagnification( void *Map );

/* adjust coordinate magnification factor of map */
int MapSetCoordinateMagnification( void *Map, long MagF );

/* request coordinate magnification factor from map */
long MapGetSoundingMagnification( void *Map );

/* adjust coordinate magnification factor of map */
int MapSetSoundingMagnification( void *Map, long MagF );
```

```
/* get projection parameters currently used */
int MapGetProjection( void *Map, int *Projection,
    double *Par1, double *Par2, double *Par3, double *Par4 );

/* query map horizontal datum */
int MapGetHorDatum( void *Map, int *Datum );

/* returns minimal lattitude or MAXLONG on error */
long MapGetMinY( void *Map );

/* returns minimal longitude or MAXLONG on error */
long MapGetMinX( void *Map );

/* returns maximal lattitude or MINLONG on error */
long MapGetMaxY( void *Map );

/* returns maximal longitude or MINLONG on error */
long MapGetMaxX( void *Map );

/*
    returns first object in specified class or
    NULL on error
*/
void *MapGetFirstObjectInClass( void *Map,
    const char ClassName[ CLASS_NAME_LEN ]);

/*
    returns next object in class
    or NULL on error or when no more objects
*/
void *MapGetNextObjectInClass( void *Map,
    const char ClassName[ CLASS_NAME_LEN ]);

/* finds object by its coordinates with tolerance Delta */
int MapGetObjByCoord( void *Map, void *MapDisp, long MagF,
    POINT *Point, long Delta, void ***Mobj );

/*
    returns first object of map or
    NULL on error
*/
void *MapGetFirstObject( void *Map );
```

```
/*
    returns next object of map ( from all classes )
    or NULL on error (or when no more objects )
*/
void *MapGetNextObject( void *Map );

/* returns minimal longitude or MAXLONG on error */
long MapObjGetMinX( void *Obj );

/* returns minimal lattitude or MAXLONG on error */
long MapObjGetMinY( void *Obj );

/* returns maximal longitude or MINLONG on error */
long MapObjGetMaxX( void *Obj );

/* returns maximal lattitude or MINLONG on error */
long MapObjGetMaxY( void *Obj );

/* return nr of attributes */
int MapObjGetAttributeCount( void *Obj );

/*
    returns the value of named attribute in Attr
    does not allocate storage
    return 0 on sucess, -1 on error
*/
int MapObjGetAttributeValue( void *Obj, ATTRIBUTE *Attr );

/*
    copies attribs of Obj to Attributes and returns number
    of attributes
    allocates storage for Attributes which must be freed by
    caller
*/
int MapObjGetAttributes( void *Obj, ATTRIBUTE **Attributes );

/*
    returns nr of points in Obj (total for all exterior and
    interior contours)
*/
int MapObjGetPointCount( void *Obj );

/*
    copies points of Obj to Points and returns nr of points
```

```
    storage is allocated
*/
int MapObjGetPoints( void *Obj,
    unsigned long *ContourCount, CONTOUR **Points );

/* returns class name of specified object */
char *MapObjGetClass( void *Obj );

/*
    returns type of specified object
    'A' - area
    'L' - line
    'P' - point
    on error '\0' is returned
*/
char MapObjGetType( void *Obj );

/*
    sets object's class, also modifies presentation
        instrucion.
    returns 0 if succesful
*/
int MapObjSetClass( void *Obj,
    const char ClassName[ CLASS_NAME_LEN ]);

/*
    sets object's points. old points are freed.
    changes Min & Max points
    returns total point count
*/
int MapObjSetPoints( void *Obj,
    unsigned long ContoursCount, CONTOUR *Contours );

/*
    sets Obj attributes from Attributes, returns Count.
    modifies presentation instruction
*/
int MapObjSetAttributes( void *Obj, int Count,
    ATTRIBUTE *Attributes );

/*
    deletes one attribute entry, returns 0 if successful
    also modifies presentation instrucion
*/
```

```
int MapObjDelAttribute( void *Obj,
    const char Name[ ATTR_NAME_LEN ]);

/*
    deletes all attributes of an object
    modifies presentation instruction
*/
int MapObjDelAttributes( void *Obj );

/* sets the value of named attribute */
int MapObjSetAttributeValue( void *Obj,
    const char Name[ ATTR_NAME_LEN ], char *Value );

/*
    create a object and add to map database
*/
void *MapObjCreate( void *Map,
    const char ClassName[ CLASS_NAME_LEN ], char Primitive,
    unsigned long ContourCount, CONTOUR *Contours,
    int AttributeCount, ATTRIBUTE *Attributes );
/*
    turn on/off highlightning for given object
*/
int MapObjHighlight( void *Obj, int Highlight );

/* frees map object */
int MapObjDestroy( void *Obj );

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

extern ERRCODE MapError;    /* last error code */

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

#endif
/*
    #ifndef MAPPER_H
*/
```

Appendix B

Sample Configuration File

This appendix lists a configuration file that is used in PPHMS. The file is built from sections, describing configuration for smaller entity. Every section is defined by appropriate header (part name in square brackets). The contents can be divided into three principal parts — general presentation parameters, list of feature object classes and then descriptions of four color schemes (each in separate section). Everything from a hash mark (#) to the end of line is ignored in parsing stage.

```
#this is a comment

[Presentation]
SymbolSet simplified
ColorScheme Day_bright

TimesFont times
UniversFont lucida

[Classes]

# CARTOGRAPHIC OBJECTS

MapClass $AREAS   1 Cartographic area
MapClass $LINES   2 Cartographic line
MapClass $CSYMB   3 Cartographic symbol
MapClass $CLOLN   4 Closing line
MapClass $COMPS   5 Compass
MapClass $NTEXT   6 National Character Set Text
MapClass $SHABL   7 Shallow water blue
MapClass $TEXTS   8 Text

# FEATURE OBJECTS
```

```
MapClass AIRARE   9 Airport area
MapClass ACHPNT  10 Anchor
MapClass ACHBRT  11 Anchor berth
MapClass ACHARE  12 Anchorage area
MapClass BCNCAR  13 Beacon , cardinal
MapClass BCNISD  14 Beacon , isolated danger
MapClass BCNLAT  15 Beacon , lateral
MapClass BCNSAW  16 Beacon , safe water
MapClass BCNSPP  17 Beacon , special purpose
MapClass BRTFAC  18 Berthing facility
MapClass BRIDGE  19 Bridge
MapClass BUIREL  20 Building , religious
MapClass BUISGL  21 Building , single
MapClass BUAARE  22 Built-up area
MapClass BOYCAR  23 Buoy , cardinal
MapClass BOYINB  24 Buoy , installation
MapClass BOYISD  25 Buoy , isolated danger
MapClass BOYLAT  26 Buoy , lateral
MapClass BOYSAW  27 Buoy , safe water
MapClass BOYSPP  28 Buoy , special purpose
MapClass CBLARE  29 Cable area
MapClass CBLOHD  30 Cable , overhead
MapClass CBLSUB  31 Cable , submarine
MapClass CAIRNS  32 Cairn
MapClass CANALS  33 Canal
MapClass CANBNK  34 Canal bank
MapClass CTSARE  35 Cargo transhipment area
MapClass CAUSWY  36 Causeway
MapClass CTNARE  37 Caution area
MapClass CEMTRY  38 Cemetery
MapClass CHNWIR  39 Chain/Wire
MapClass CHKPNT  40 Checkpoint
MapClass CHIMNY  41 Chimney
MapClass CGUSTA  42 Coastguard station
MapClass COALNE  43 Coastline
MapClass CONZNE  44 Contiguous zone
MapClass COSARE  45 Continental shelf area
MapClass CTRPNT  46 Control point
MapClass CRANES  47 Crane
MapClass CUSZNE  48 Custom zone
MapClass DAMCON  49 Dam
MapClass DWRTCL  50 Deep water route centreline
MapClass DWRTPT  51 Deep water route part
```

```
MapClass DEPARE  52 Depth area
MapClass DEPCNT  53 Depth contour
MapClass DIFFUS  54 Diffuser
MapClass DSHAER  55 Dish aerial
MapClass DISMAR  56 Distance mark
MapClass DOCARE  57 Dock area
MapClass DRGARE  58 Dredged area
MapClass DRYDOC  59 Dry dock
MapClass DMPGRD  60 Dumping ground
MapClass DUNARE  61 Dune
MapClass DYKARE  62 Dyke area
MapClass DYKCRW  63 Dyke crown
MapClass EXEZNE  64 Exclusive economic zone
MapClass FAIRWY  65 Fairway
MapClass FNCLNE  66 Fence
MapClass FERYRT  67 Ferry route
MapClass FSHHAV  68 Fish haven
MapClass FSHZNE  69 Fishery zone
MapClass FSHFAC  70 Fishing facility
MapClass FSHGRD  71 Fishing ground
MapClass FLGSTF  72 Flagstaff/Flagpole
MapClass FLASTK  73 Flare stack
MapClass FLODOC  74 Floating dock
MapClass FOGSIG  75 Fog signal
MapClass FORSTC  76 Fortified structure
MapClass FRPARE  77 Free port area
MapClass GATCON  78 Gate
MapClass GRIDRN  79 Gridiron
MapClass HRBARE  80 Harbour area (administrative)
MapClass HRBFAC  81 Harbour facility
MapClass HILARE  82 Hill
MapClass HULKES  83 Hulk
MapClass ICEARE  84 Ice area
MapClass ICNARE  85 Incineration area
MapClass ITDARE  86 Intertidal area
MapClass LAKARE  87 Lake
MapClass LAKSHR  88 Lake - shore
MapClass LNDARE  89 Land area
MapClass LNDELV  90 Land elevation
MapClass LNDRGN  91 Land region
MapClass LNDPLC  92 Landing place
MapClass LNDSTS  93 Landing stairs
MapClass LIGHTS  94 Light
MapClass LITFLT  95 Light float
```

```
MapClass LITMOI  96 Light , moiré effect
MapClass LITVES  97 Light vessel
MapClass LOCMAG  98 Local magnetic anomaly
MapClass LOKBSN  99 Lock basin
MapClass LOGPON 100 Log pond
MapClass MARCUL 101 Marine farm/culture
MapClass MSTCON 102 Mast
MapClass MIPARE 103 Military practice area
MapClass MONUMT 104 Monument
MapClass MORFAC 105 Mooring/Warping facility
MapClass NATARE 106 National territorial area
MapClass NAVLNE 107 Navigation line
MapClass OBSTRN 108 Obstruction
MapClass OFSPLF 109 Offshore platform
MapClass OFSPRD 110 Offshore production area
MapClass OILBAR 111 Oil barrier
MapClass PILPNT 112 Pile
MapClass PILBOP 113 Pilot boarding place
MapClass PINGOS 114 Pingo
MapClass PIPARE 115 Pipeline area
MapClass PIPOHD 116 Pipeline , overhead
MapClass PIPSOL 117 Pipeline , submarine/on land
MapClass PONTON 118 Pontoon
MapClass PRCARE 119 Precautionary area
MapClass PRDINS 120 Production installation
MapClass PYLONS 121 Pylon
MapClass RADDOM 122 Radar dome
MapClass RADLNE 123 Radar line
MapClass RADRNG 124 Radar range
MapClass RADRFL 125 Radar reflector
MapClass RADSTA 126 Radar station
MapClass RTPBCN 127 Radar transponder beacon
MapClass RDOCAL 128 Radio calling-in point
MapClass RDOSTA 129 Radio station
MapClass RAILWY 130 Railway
MapClass RMPARE 131 Ramp
MapClass RAPIDS 132 Rapids
MapClass RCRTCL 133 Recommended route centerline
MapClass RECTRC 134 Recommended track
MapClass RCTLPT 135 Recommended traffic lane part
MapClass RSCSTA 136 Rescue station
MapClass RESARE 137 Restricted area
MapClass RIVERS 138 River
MapClass RIVBNK 139 River bank
```

```
MapClass RODCRS 140 Road crossing
MapClass ROADPT 141 Road part
MapClass RUNWAY 142 Runway
MapClass SLTPAN 143 Salt pan
MapClass SNDWAV 144 Sand waves
MapClass SEAARE 145 Sea area
MapClass SPLARE 146 Sea-plane landing area
MapClass SBDARE 147 Seabed area
MapClass SLCONS 148 Shoreline construction
MapClass SISTAT 149 Signal station , traffic
MapClass SISTAW 150 Signal station , warning
MapClass SILBUI 151 Silo
MapClass SLIPWY 152 Slipway
MapClass SLOTOP 153 Slope topline
MapClass SLOGRD 154 Sloping ground
MapClass SMCFAC 155 Small craft facility
MapClass SOUNDG 156 Sounding
MapClass SPOGRD 157 Spoil ground
MapClass SPRING 158 Spring
MapClass SQUARE 159 Square
MapClass STSLNE 160 Straight territorial sea baseline
MapClass SUBTLN 161 Submarine transit lane
MapClass TNKCON 162 Tank
MapClass TELPHC 163 Telepheric
MapClass TESARE 164 Territorial sea area
MapClass TIDEWY 165 Tideway
MapClass TOPMAR 166 Top mark
MapClass TOWERS 167 Tower
MapClass TSELNE 168 Traffic separation line
MapClass TSSBND 169 Traffic separation scheme boundary
MapClass TSSCRS 170 Traffic separation scheme crossing
MapClass TSSLPT 171 Traffic separation scheme lane part
MapClass TSSRON 172 Traffic separation scheme roundabout
MapClass TSEZNE 173 Traffic separation zone
MapClass TREPNT 174 Tree
MapClass TNLENT 175 Tunnel entrance
MapClass TWRTPT 176 Two-way route part
MapClass UWTROC 177 Underwater rock
MapClass VEGARE 178 Vegetation area
MapClass WATFAL 179 Waterfall
MapClass WATTUR 180 Water turbulence
MapClass WEDKLP 181 Weed/Kelp
MapClass WIRLNE 182 Weir
MapClass WNDMIL 183 Windmill
```

```
MapClass WIMCON 184 Windmotor
MapClass WRECKS 185 Wreck
MapClass ZEMCNT 186 Zero meter contour

# META OBJECTS

MapClass M_ACC 187 Accuracy of data
MapClass M_CSC 188 Compilation scale of dat
MapClass M_HDA 189 Horizontal scale of dat
MapClass M_NPU 190 Nautical publication informati
MapClass M_PRO 191 Production information
MapClass M_SDA 192 Sounding datum of data
MapClass M_SRE 193 Survey reliability
MapClass M_SSO 194 Survey source
MapClass M_UNI 195 Units of measurement
MapClass M_VDA 196 Vertical datum of data

#color_name (case insensitive) red green blue (in hexadecimal)

[DAY_BRIGHT]

MapColor nodta a699 8e38    79e7
MapColor cursr ffff 69a6    1040
MapColor chblk 0000 0000    0000
MapColor chgrd a699 8e38    79e7
MapColor chgrf d75c baea    a699
MapColor chred ffff 5144    6185
MapColor chgrn 0000 efbe    28a2
MapColor chylw ffff db6c    28a2
MapColor chmgd ffff 1c71    befb
MapColor chmgf ffff a699    c71b
MapColor chbrn ffff cb2b    69a6
MapColor chwht ffff ffff    df7d
MapColor sclbr ffff 69a6    1040
MapColor chcor ffff 69a6    1040
MapColor isdng 0000 8207    cf3c
MapColor dnghl ffff 4103    5965
MapColor trfcd ffff 1c71    befb
MapColor trfcf ffff a699    c71b
MapColor landa ffff df7d    8617
MapColor landf c71b 69a6    1040
MapColor cstln 0000 0000    0000
MapColor sndg1 a699 8e38    79e7
MapColor sndg2 0000 0000    0000
```

```
MapColor depsc a699 8e38    79e7
MapColor depcn a699 8a28    79e7
MapColor depdw ffff f3ce    cf3c
MapColor depmd ffff f3ce    d75c
MapColor depms c71b efbe    cf3c
MapColor depvs aeba e79d    cf3c
MapColor depit df7d e79d    a699
MapColor radhi 0000 aeba    5144
MapColor radlo 0000 f3ce    69a6
MapColor resbl 0000 8207    cf3c
MapColor resyw ffff db6c    28a2
MapColor resgr 4924 4d34    5144
MapColor ships 0000 0000    0000
MapColor plrte ffff 0000    0000
MapColor aplrt ffff 69a6    1040
MapColor outlw 0000 0000    0000
MapColor outll b6da aaaa    69a6
MapColor uinfd 0000 0000    0000
MapColor uinff cf3c cf3c    cf3c
MapColor uibck ffff ffff    ffff
MapColor uiafd 0000 0000    cf3c
MapColor uinfr ffff 0000    0000
MapColor uinfg 0000 ffff    0000
MapColor uinfo ffff a699    0000
MapColor uinfb 0000 0000    ffff
MapColor uinfm ffff 0000    ffff
MapColor uibdr b6da b6da    b6da
MapColor uiaff cf3c 8617    38e3
# !!!!!

[DAY_BLACKBACK]

MapColor nodta a699 8e38    79e7
MapColor cursr ffff 69a6    1040
MapColor chblk 0000 0000    0000
MapColor chgrd a699 8e38    79e7
MapColor chgrf d75c baea    a699
MapColor chred ffff 5144    6185
MapColor chgrn 0000 efbe    28a2
MapColor chylw ffff db6c    28a2
MapColor chmgd ffff 1c71    befb
MapColor chmgf ffff a699    c71b
MapColor chbrn ffff cb2b    69a6
MapColor chwht ffff ffff    df7d
```

```
MapColor sclbr ffff 69a6    1040
MapColor chcor ffff 69a6    1040
MapColor isdng 0000 8207    cf3c
MapColor dnghl ffff 4103    5965
MapColor trfcd ffff 1c71    befb
MapColor trfcf ffff a699    c71b
MapColor landa ffff df7d    8617
MapColor landf c71b 69a6    1040
MapColor cstln 0000 0000    0000
MapColor sndg1 a699 8e38    79e7
MapColor sndg2 0000 0000    0000
MapColor depsc a699 8e38    79e7
MapColor depcn a699 8a28    79e7
MapColor depdw ffff f3ce    cf3c
MapColor depmd ffff f3ce    d75c
MapColor depms c71b efbe    cf3c
MapColor depvs aeba e79d    cf3c
MapColor depit df7d e79d    a699
MapColor radhi 0000 aeba    5144
MapColor radlo 0000 f3ce    69a6
MapColor resbl 0000 8207    cf3c
MapColor resyw ffff db6c    28a2
MapColor resgr 4924 4d34    5144
MapColor ships 0000 0000    0000
MapColor plrte ffff 0000    0000
MapColor aplrt ffff 69a6    1040
MapColor outlw 0000 0000    0000
MapColor outll b6da aaaa    69a6
MapColor uinfd 0000 0000    0000
MapColor uinff cf3c cf3c    cf3c
MapColor uibck ffff ffff    ffff
MapColor uiafd 0000 0000    cf3c
MapColor uinfr ffff 0000    0000
MapColor uinfg 0000 ffff    0000
MapColor uinfo ffff a699    0000
MapColor uinfb 0000 0000    ffff
MapColor uinfm ffff 0000    ffff
MapColor uibdr 4924 4d34    4924
MapColor uiaff cf3c 8617    38e3
# !!!

[DUSK_BLACKBACK]

MapColor nodta 0000 0820    0000
```

```
MapColor cursr 79e7 28a2    0000
MapColor chblk 0000 0000    0000
MapColor chgrd 4103 2cb2    28a2
MapColor chgrf 5965 4103    38e3
MapColor chred 79e7 1c71    0000
MapColor chgrn 0000 5144    0000
MapColor chylw 71c6 4d34    0000
MapColor chmgd 69a6 0410    4103
MapColor chmgf 71c6 34d3    4924
MapColor chbrn 69a6 4514    28a2
MapColor chwht 71c6 5144    5144
MapColor sclbr 79e7 28a2    0000
MapColor chcor 79e7 28a2    0000
MapColor isdng 0000 2cb2    4924
MapColor dnghl 79e7 1c71    0000
MapColor trfcd 69a6 0410    4103
MapColor trfcf 71c6 34d3    4924
MapColor landa 69a6 4d34    30c2
MapColor landf 5144 2492    0000
MapColor cstln 0000 0000    0000
MapColor sndg1 4924 30c2    30c2
MapColor sndg2 5965 4103    38e3
MapColor depsc 4924 30c2    30c2
MapColor depcn 4924 30c2    30c2
MapColor depdw 69a6 5555    4924
MapColor depmd 5965 5555    5144
MapColor depms 5965 5144    5144
MapColor depvs 5965 5144    5144
MapColor depit 5144 5144    38e3
MapColor radhi 0000 30c2    0000
MapColor radlo 0000 4d34    28a2
MapColor resbl 0000 2cb2    4924
MapColor resyw 71c6 4d34    0000
MapColor resgr 5965 4103    38e3
MapColor ships 0000 0000    0000
MapColor plrte 79e7 0000    0000
MapColor aplrt 79e7 28a2    0000
MapColor outlw 0000 0000    0000
MapColor outll b6da aaaa    69a6
MapColor uinfd 0000 0000    0000
MapColor uinff cf3c cf3c    cf3c
MapColor uibck ffff ffff    ffff
MapColor uiafd 0000 0000    cf3c
MapColor uinfr ffff 0000    0000
```

```
MapColor uinfg 0000 ffff    0000
MapColor uinfo ffff a699    0000
MapColor uinfb 0000 0000    ffff
MapColor uinfm ffff 0000    ffff
MapColor uibdr 4924 4d34    4924
MapColor uiaff cf3c 8617    38e3
#

[NIGHT_FILTERED]

MapColor nodta 5965 4d34    5144
MapColor cursr ffff 6185    1040
MapColor chblk c71b befb    befb
MapColor chgrd 5965 4d34    5144
MapColor chgrf 5965 4d34    5144
MapColor chred f7de 0000    0000
MapColor chgrn 0000 c71b    4924
MapColor chylw ffff b2ca    1861
MapColor chmgd c71b 1c71    9658
MapColor chmgf 5144  410    38e3
MapColor chbrn 4103 28a2    28a2
MapColor chwht d75c baea    a699
MapColor sclbr ffff 6595    1861
MapColor chcor ffff 6595    1861
MapColor isdng 0000 71c6    aeba
MapColor dnghl f7de 0000    0000
MapColor trfcd 8617 1040    6185
MapColor trfcf 5144  410    38e3
MapColor landa 5144 34d3    28a2
MapColor landf 5965 1861    0000
MapColor cstln c71b befb    befb
MapColor sndg1 8617 6595    69a6
MapColor sndg2 69a6 4d34    5144
MapColor depsc 8617 6595    69a6
MapColor depcn 8617 6595    69a6
MapColor depdw 0000 0000    0000
MapColor depmd 0000 0000    1861
MapColor depms 0000 0410    4103
MapColor depvs 0000 0820    5144
MapColor depit 0000 28a2    28a2
MapColor radhi 0000 8617    38e3
MapColor radlo 0000 5144    2081
MapColor resbl 0000 71c6    aeba
MapColor resyw ffff b2ca    1861
```

```
MapColor resgr 5965 4d34    5144
MapColor ships d75c baea    befb
MapColor plrte b6da 0000    0000
MapColor aplrt ffff 6595    2081
MapColor outlw 0000 0000    0000
MapColor outll b6da aaaa    69a6
MapColor uinfd 0000 0000    0000
MapColor uinff cf3c cf3c    cf3c
MapColor uibck ffff ffff    ffff
MapColor uiafd 0000 0000    cf3c
MapColor uinfr ffff 0000    0000
MapColor uinfg 0000 ffff    0000
MapColor uinfo ffff a699    0000
MapColor uinfb 0000 0000    ffff
MapColor uinfm ffff 0000    ffff
MapColor uibdr 4924 4d34    4924
MapColor uiaff cf3c 8617    38e3
#end
```