Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1988-03

A standard library for modeling satellite orbits on a microcomputer

Beutel, Kenneth L.

Monterey, California. Naval Postgraduate School

https://hdl.handle.net/10945/23404

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

DUDLEY KNOX LIBRARY
SCHOOL
93943-5002

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

B435

A STANDARD LIBRARY FOR MODELING

SATELLITE ORBITS ON A MICROCOMPUTER

by

Kenneth L. Beutel

March 1988

Thesis Advisor: D. Davis
Co-Advisor: D. Boger

Approved for public release; distribution is unlimited.

T238725

nclassified

urity classification of this page

REPORT DOCUMENTATION PAGE

Report Security Classification Unclassified		1b Restrictive Markings
Security Classification Authority		3 Distribution Availability of Report Approved for public release; distribution is unlimited.
Declassification Downgrading Schedule		
Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)
Name of Performing Organization aval Postgraduate School	6b Office Symbol *(if applicable)* 54	7a Name of Monitoring Organization Naval Postgraduate School
Address *(city, state, and ZIP code)* lonterey, CA 93943-5000		7b Address *(city, state, and ZIP code)* Monterey, CA 93943-5000
Name of Funding Sponsoring Organization	8b Office Symbol *(if applicable)*	9 Procurement Instrument Identification Number
Address *(city, state, and ZIP code)*		10 Source of Funding Numbers Program Element No | Project No | Task No | Work Unit Accession No

Title *(include security classification)* A STANDARD LIBRARY FOR MODELING SATELLITE ORBITS ON A MICRO-OMPUTER

Personal Author(s) Beutel, Kenneth L.

Type of Report	13b Time Covered	14 Date of Report *(year, month, day)*	15 Page Count
laster's Thesis	From To	1988 March	239

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

Cosati Codes			18 Subject Terms *(continue on reverse if necessary and identify by block number)*
eld	Group	Subgroup	orbital mechanics, computer programming

Abstract *(continue on reverse if necessary and identify by block number)*

troductory students of astrodynamics and the space environment are required to have a fundamental understanding for the nematic behavior of satellite orbits. This thesis develops a standard library that contains the basic formulas for modeling rth orbiting satellites. This library is used as a basis for implementing a satellite motion simulator that can be used to monstrate orbital phenomena in the classroom. This thesis surveys the equations and principles taught to introductory bital mechanics and space systems students. The library organizes these orbital elements, coordinate systems and analytic mulas into a standard method for modeling earth orbiting satellites. The standard library is written in the C programming nguage and is designed to be highly portable between a variety of computer environments. The simulation draws heavily the standards established by the library to produce a graphics-based orbit simulation program written for the Apple acintosh computer. The simulation demonstrates the utility of the standard library functions but, because of its extensive e of the Macintosh user interface, is not portable to other operating systems.

Distribution Availability of Abstract unclassified unlimited ☐ same as report ☐ DTIC users	21 Abstract Security Classification Unclassified	
Name of Responsible Individual C. Boger	22b Telephone *(include Area code)* (408) 646-2607	22c Office Symbol 54Bo

FORM 1473,84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

security classification of this page

Unclassified

Approved for public release; distribution is unlimited.

A Standard Library for Modeling Satellite Orbits on a Microcomputer

by

Kenneth L. Beutel
Captain, United States Marine Corps
B.S., State University of New York College at Buffalo, 1981

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY
(SPACE SYSTEMS OPERATIONS)

and

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1988

ABSTRACT

Introductory students of astrodynamics and the space environment are required to have a fundamental understanding for the kinematic behavior of satellite orbits. This thesis develops a standard library that contains the basic formulas for modeling earth orbiting satellites. This library is used as a basis for implementing a satellite motion simulator that can be used to demonstrate orbital phenomena in the classroom.

This thesis surveys the equations and principles taught to introductory orbital mechanics and space systems students. The library organizes these orbital elements, coordinate systems and analytic formulas into a standard method for modeling earth orbiting satellites. The standard library is written in the C programming language and is designed to be highly portable between a variety of computer environments.

The simulation draws heavily on the standards established by the library to produce a graphics-based orbit simulation program written for the Apple Macintosh computer. The simulation demonstrates the utility of the standard library functions but, because of its extensive use of the Macintosh user interface, is not portable to other operating systems.

Thesis
B455
c.1

TABLE OF CONTENTS

I. INTRODUCTION 1
A. METHODOLOGY 2
B. ENVIRONMENT 3
II. BACKGROUND 4
A. MOTIVATION 4
B. PHILOSOPHY 5
C. DESIGN DECISIONS 6
III. EQUATIONS OF ORBITAL MOTION 8
A. THE TWO BODY CLOSED ORBIT PROBLEM 8
1. Assumptions 8
2. The Four Basic Laws of Satellite Motion 9
B. ORBITAL EQUATIONS 10
1. The Equations of an Ellipse 10
2. Orbital Constants 11
3. Relationships Between the Constants of the Motion and Geometry 13
4. Earth Coverage and Line of Sight Distance to the Horizon 14
C. TIME 16
1. Solar Time 16
2. Sidereal Time 17
D. CANONICAL UNITS 18
E. CORRECTION FOR ASPHERICAL GRAVITATIONAL POTENTIAL 19
IV. CELESTIAL COORDINATE SYSTEMS AND TRANSFORMATIONS 21
A. GEOCENTRIC-EQUATORIAL (**IJK**) COORDINATE SYSTEM 21
B. RIGHT ASCENSION-DECLINATION (**R.A.-DECL**) COORDINATE SYSTEM 22
C. GEOGRAPHIC ($\phi\lambda h$) COORDINATE SYSTEM 23

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

D. TOPOCENTRIC (SEZ) COORDINATE SYSTEM 25
E. PERIFOCAL (PQW) COORDINATE SYSTEM 26
F. KEPLERIAN ELEMENTS 27
G. TRANSFORMATION FORMULAS 30
1. Keplerian to **PQW** 30
2. **PQW** and Keplerian to **IJK** 31
3. **IJK** and Keplerian to **PQW** 31
4. **SEZ** to **IJK** 31
5. Right Ascension-Declination to **IJK** 32
6. **IJK** to Right Ascension-Declination 32
7. **IJK** to Geographic ($\phi\lambda h$) 32
8. Geographic to **IJK** 33
V. ORBIT PREDICTION 34
A. STATEMENT OF THE PROBLEM 34
B. OUTLINE OF THE SOLUTION 34
1. Compute the Mean Anomaly (M) at Time t 35
2. Determine the Eccentric Anomaly from the Mean Anomaly 37
3. Obtain the Fosition Vector in the Perifocal (PQW) Coordintate System 37
4. Transform the Result into the Coordinate System of Choice 38
VI. COMMON ORBIT CLASSIFICATIONS 39
A. ORBIT CLASSIFICATION BY ALTITUDE 39
1. Low Earth Orbit (LEO) 39
2. High Earth Orbit (HEO) 40
B. ORBIT CLASSIFICATION BY INCLINATION 40
1. Sun Synchronous Orbit 40
2. Molniya Orbit 41
C. ORBIT CLASSIFICATION BY PERIOD 42
1. Geosynchronous Orbit (P=23hr. 56min.) 42
2. Semi-synchronous Orbit (P=11hr. 58min.) 42
VII. IMPLEMENTATION 44
A. INTRODUCTION 44
B. LIBRARY NAMING CONVENTIONS 45

C. THE GENERAL LIBRARY 46
D. THE CONVERSION LIBRARY 46
E. THE TIME LIBRARY..... 47
F. THE COORDINATE SYSTEM LIBRARY..... 48
G. THE KEPLER PROBLEM LIBRARY 49
VIII. CONCLUSIONS AND RECOMMENDATIONS 50
APPENDIX A SYMBOL GLOSSARY 52
APPENDIX B STANDARD LIBRARY SOURCE LISTING 56
APPENDIX C STANDARD LIBRARY DOCUMENTATION 88
APPENDIX D MACORBITS SOURCE LISTING 116
APPENDIX E MACORBITS USER'S GUIDE 213

LIST OF FIGURES

Figure 1 The Ellipse 10
Figure 2 Relationship of the Flight Path Angle to Position (r)
and Velocity (v) Vectors at Two Selected Locations 13
Figure 3 Line of Sight Distance (ρ) to the Horizon 15
Figure 4 Geocentric-Equatorial Coordinate System 22
Figure 5 Right Ascension-Declination Coordinate System 23
Figure 6 Geographic Coordinate System 25
Figure 7 Topocentric Coordinate System 26
Figure 8 Perifocal Coordinate System 27
Figure 9 Keplerian Elements 28
Figure 10 True Anomaly (ν) and Eccentric Anomaly (E) Relationship 29
Figure 11 First Order Perturbative Effects 36

LIST OF TRADEMARKS

Apple is a Registered Trademark of Apple Computer, Inc.

Macintosh is a Trademark of McIntosh Laboratory, Inc. and is used by Apple Computer, Inc. with its express permission

THINK'S LightSpeedC is a Copyright of Think Technologies, Inc.

UNIX is a registered trademark of AT&T Bell Laboratories Inc.

I. INTRODUCTION

Most students of astrodynamics, that branch of mechanics that is concerned with the motion of celestial bodies and often called celestial mechanics, are given a rigorous treatment of the physics but little appreciation for the actual kinematic behavior of a satellite's orbit. The reasons for this are twofold; the first is that the equations involved are often time dependent functions. Secondly, and perhaps more importantly, it is not an easy task to represent a three dimensional position in a single geometric plane such as the classroom blackboard. These two factors, and the requirement to perform repetitive, complex calculations rapidly, are enough justification for developing a standard orbit modeling library on a microcomputer system.

This standard library should provide facilities to support the creation and calculation of orbits from simple elementary cases to more advanced modeling of actual Earth satellite systems. Based on this library an interactive orbital simulation can be created to give the student of astrodynamics an appreciation for exactly what characterizes the motion of a satellite in orbit around the Earth. Unfortunately, most precise orbit modeling systems are not presented as pedagogical tools or require special hardware that make them unavailable for use on microcomputers. The systems that are available for microcomputers are often monolithic structures that can provide the student with little insight into the particular components of an orbital simulation. Of particular importance to a simulation designed to educate the user is the ability for the student to interact with the information presented. This requires an integrated system capable of providing immediate feedback and easy modification of the parameters that define a satellite orbit.

The purpose of this thesis is to provide a student with an accessible resource for the study of satellite orbits. This is accomplished by first creating a library of useful and independent routines that can be used in many potential orbit modeling situations. The utility of these modules is subsequently demonstrated by using them to create a multiple view, three dimensional simulation of several satellites orbiting the Earth. To make these modules truly general purpose, the C programming language was chosen as the only language currently in widespread use that is capable of fast numeric calculations and is available in several different microcomputer operating systems and graphics environments. The C programming language has the added benefit of being able to provide a structured interface that allows many of the implementation details of the orbital routine library to be abstracted out. Such abstraction will make future enhancements or maintenance as painless as possible to the user programs serviced by the library.

A. METHODOLOGY

This thesis is organized into three main sections: a background of relevant astrodynamic concepts, a description of library functions, and the orbital simulation program. The section devoted to providing background material includes chapters that discuss the motion of satellites in a closed orbit and the coordinate systems that this motion can be presented in. By stepwise development of the computational aspects of astrodynamics, an appreciation for the function performed by a particular library routine as well as its context can be established.

The simulation will make use of many of the library routines developed but is not intended to encompass the entire library. Some routines are provided for completeness and would be used by other programs for their own specific objectives. Finally, all routines included in the standard library are documented in the UNIX style (i.e., individually or grouped logically one per page specifying data types and return values).

B. ENVIRONMENT

As stated earlier the library is designed to be as portable as possible, but the simulation was programmed with a specific family of machines in mind; the Apple Macintosh. These computers, especially the latest introduction -- the Macintosh II, represent of the current state of the art in microcomputers. The Macintosh II is a high resolution color system, with a 32-bit microprocessor (Motorola 68020), arithmetic coprocessor (Motorola 60881) and a mature graphics-based operating system. Although the simulation will execute on earlier models of the Macintosh, certain operations (such as floating point calculations) will realize an increase in speed when performed on the Macintosh II.

The software was compiled using Think Technologies' LightSpeed C compiler environment. This version of the C programming language provides a standard C implementation that allows the library to be machine portable. However, the simulation program is machine specific because of its extensive use of the Macintosh user interface including resources, event management, and drawing routines. The numeric values produced by the standard library are scaled and processed by the simulation to produce results visible on the graphics display.

II. BACKGROUND

A. MOTIVATION

Communicating information about a satellite's orbit is difficult because the solution requires visualizing a three dimensional coordinate system, the dynamic motion of satellite, the rotation of the Earth, and some appreciation for the long term changes that the orbit is experiencing. It would be worthwhile to attempt to isolate individual aspects of the orbit for study and gradually add some new characteristic to the problem. This representation of orbital motion can be accomplished by physical models, using motors or lights [Ref. 1:pp.16-18] or the physical model itself can be represented by computer software. There are advantages to both physical and software modeling techniques.

There is some loss of spatial information in the software model because of the two dimensional nature of a computer's display screen. The viewer is restricted to the view angles and perspective offered by the software system and is further limited by screen resolution, object lighting and shading models offered. To offer more vantage points or increase realism in the display is taxing to the computer's central processing unit and may slow the model so that it cannot hold the attention of the viewer for a long period of time.

Fortunately, reality does not need to be modeled precisely if the user is provided a reasonable representation for the essential aspects of the orbit. If this abstraction is acceptable then there exist many advantages of the software model over the hardware implementation. The first and foremost advantage is that the algorithms, once encapsulated in software, are applicable to the entire class of orbits. Thus, while the hardware designer is still bending wires and changing gear ratios the software implementation, after a few changes to variables, is already running.

Other reasons for adopting the software model include lower cost, lack of mechanical problems that are common to custom built hardware, the ability of software to preserve its current status for later viewing, and the reusability or extendability of the software for other projects. Although the physical model is useful in some circumstances (such as science exhibits) it is not the model of choice for an interactive demonstrator, comparing different orbits, or for observing the effects of small changes to the current orbit under study.

B. PHILOSOPHY

The software model chosen to represent the motion of the satellite is based on several design goals. These goals are formulated to meet the needs of the principle user, the early student of astrodynamics. Therefore the output produced by the model and even the methods used to obtain the output closely parallel the material covered during a first course in orbital mechanics.

The software model provided consists of two complementary parts: the standard library and the simulation. The standard library is a collection of routines unified by common data structures that provide support for several independent functions related to the study of orbits. The simulation is a Macintosh-specific application that demonstrates the use of the standard library in the context of a major program devoted to the modeling of near earth satellites.

Frequently, the methods and coordinates used by the standard library are the historical or classic approaches to the problem and have been superseded by more modern computationally intensive techniques. The standard library is based on classical astrodynamic techniques because the newer methods are not generally taught in introductory courses and the student is expected to learn by examining how the standard library operates.

The design philosophy for the standard library is carried over into the simulation program. Specifically the simulation is constructed so that the program would allow:

1. Multiple views of the same satellite, that allow viewing the satellite in the orbital plane while it also traces out the ground track on a mercator projection of the Earth.
2. The representation of multiple satellites simultaneously for the purposes of comparison of their individual characteristics.
3. The storage and retrieval of user defined orbits, as well as provide "canned" sample orbit parameters.
4. The modification of both measurement systems and time scales for viewing purposes.

C. DESIGN DECISIONS

Although it would be ideal for the standard library and simulation to include code to handle every possible user's requirements, it is not realistic to expect that this can be done. From the outset, the standard library was designed to meet the minimum needs of a user being introduced to the computation of satellite orbits by obtaining data points for a selected set of coordinate systems. In keeping with the standard library 's pedagogical flavor, only truly universal constants are hard coded into the software. In other words, the user is free to experiment with planetary systems of different masses, shapes and sizes than the values commonly used for the Earth.

Simplifications to both the orbital model and software library were necessary before the actual coding of the software could begin. Because of the difficulty in analytically modeling certain orbital perturbations and the need to establish a practical limit on the involvement of minor effects, only the largest perturbative effect was included. While this simplification restricts the accuracy of the calculations to the first few orbits it does not violate the philosophy behind this software system. The decision was also made to exclude the drawing routines used by the simulation from the standard library because a large percentage of these routines are machine specific.

Additionally, other computer systems may possess commercial three-dimensional drawing packages capable of more flexibility and speed than the methods used in the simulation. By acknowledging the constraints imposed on the library before the actual programming began, it is hoped that these design decisions will enhance the utility of the final product rather than allowing inconsistencies to develop between the design and the implementation.

III. EQUATIONS OF ORBITAL MOTION

The standard library was developed based on a specific model for the motion of earth orbiting satellites. This model is called the *two-body closed orbit problem* and contains many of the standard simplifying assumptions used by other simulations. This chapter describes these assumptions and then develops the two body closed orbit equations and associated constants of motion. Finally, a method for modeling the most common perturbation to the two body orbit problem, the Earth's aspherical gravitational potential, is provided.

A. THE TWO BODY CLOSED ORBIT PROBLEM

1. Assumptions

The easiest orbital motion to visualize is that of a single small object rotating around a heavier body. The more massive body is called the *primary* and the orbiting object is referred to as the *secondary*. This model is a fair representation for the motion of artificial satellites around the Earth and is composed of several simplifying assumptions. The most important assumptions are:

- The secondary body has zero mass, and thus has no gravitational attraction on the primary [Ref 2:pp. 14].
- Both bodies are perfectly spherical masses, and thus can be approximated as a single point, located at their centers [Ref 2:pp.11].
- There are no external forces acting on the two body system and the only internal force present is gravitational force [Ref 2:pp.12].

While these assumptions sound as though the final model is far removed from the "real world", in actuality the results of the model can be close to empirically observed values. For example the mass of the secondary body (the satellite) is often insignificant in

comparison with the mass of the Earth. The Space Transportation System ("space shuttle") Orbiter, which is a large satellite, has a mass of 75,000 kilograms [Ref 3:pp. 13.6] but still represents only a small fraction ($7.5 \times 10^4 \div 5.98 \times 10^{24} = 1.3 \times 10^{-20}$) of the Earth's total mass. For earth orbits with altitudes of less than 36,000 kilometers and greater than 200 kilometers most external forces are small. The most important perturbing force in that range of altitudes is the effect of the oblate Earth. Because the Earth is not a perfect sphere, the magnitude of the Earth's gravitational potential changes as the satellite changes position. This phenomenon can be added to the basic model later but provides little additional insight into the fundamental problem of characterizing the motion of an earth orbiting satellite.

2. The Four Basic Laws of Satellite Motion

Within the framework of the simplifying assumptions there are four equations from which most characteristic equations and constants are derived. The first three were developed by Johann Kepler from observations about the manner that planetary orbits behaved. These relationships are often called Kepler's Three Laws and are expressed as:

First Law - The orbit of each planet is an ellipse with the sun at a focus.

Second Law - The line joining the planet to the sun sweeps out equal areas in equal time.

Third Law - The square of the period of a planet is proportional to the cube of its mean distance from the sun. [Ref 2:pp. 2]

The fourth basic law used to model orbital behavior is Sir Isaac Newton's law of universal gravitation, which states that two bodies have a mutual attractive force that is proportional to their mass product. This law further states that the magnitude of this force is inversely proportional to the square of the distance separating the objects. Obviously other physical laws, such as Newton's three laws of motion, must also be considered but these four are of special importance when modeling orbital motion. [Ref. 2:pp. 4]

B. ORBITAL EQUATIONS

1. The Equations of an Ellipse

An ellipse is a closed curve that is one member of a family of curves called *conic sections*. A conic section is a curve that is created by slicing a plane through a hollow right circular cone at some angle [Ref. 3:pp. 2.11]. The ellipse is characterized by tracing a curve of constant distance from two points called foci (F and F'). The distance from one end of the ellipse to the other through the foci is called the major axis. The axis that is at right angles to the major axis and passes through the center of the ellipse is called the minor axis. There exists another axis perpendicular to the major axis, the *latus rectum*, which passes through the focus containing the primary. The latus rectum, major and minor axes are often referred to by their half length distances as the semi-latus rectum, semi-major (a) and semi-minor (b) axes, respectively. These relationships are shown in Figure 1. [Ref. 3:pp. 2.13]

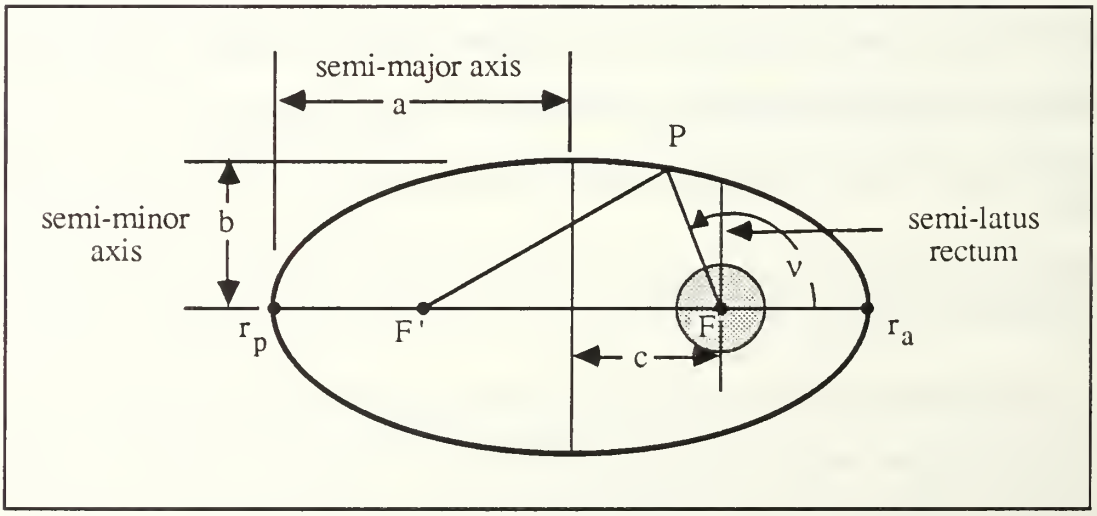


Figure 1. The Ellipse

From this figure we can establish that $a^2 = b^2 + c^2$ and define a new quantity called eccentricity as $e = c \div a$. For an ellipse the value of eccentricity can range from zero (a circle is a degenerate ellipse) and up to but not including one. [Ref. 3:pp. 2.13]

The point at the end of the major axis nearest the primary (F) is in general called the periapsis (perigee). The one at the greatest distance away from the primary is called the apoapsis (apogee). The eccentricity, semi-major axis, radius of perigee (r_p) and radius of apogee (r_a) are related by:

$r_p = a\ (1-e)$ and $r_a = a\ (1+e)$. [Ref. 2:pp. 24-25]

It is also useful to know the distance of the satellite from the primary at any point along its elliptical orbit. Figure 1 shows how the polar angle (ν) (measured counter-clockwise from perigee) can be used to locate the satellite's position along the ellipse. This angle determines the radial distance from the primary to the ellipse by [Ref. 4:pp. 82]:

$r = p \div (1 + e \cos \nu)$, where $p = a\ (1- e^2)$ is called the *semi-parameter of a conic.*

This distance may be expresses as two Cartesian coordinates (discussed in the next chapter) by [Ref. 4:pp. 82]:

$x_\omega = (p \cos \nu) \div (1 + e \cos \nu)$

$y_\omega = (p \sin \nu) \div (1 + e \cos \nu)$

2. Orbital Constants

Leaving the geometric interpretation of a satellite's orbit for the moment, there are several constants that describe important physical properties of the satellite. These properties are often called *constants of the motion* .

The force that the primary exerts on the orbiting object is manifested as the gravitational potential. For the restricted two body problem, where the mass of the satellite is essential zero, the universal gravitational constant (G) and the Earth's mass (M_{earth}) are often combined into a single constant ($\mu = GM_{earth} = 3.986012 \times 10^5$ km^3/sec^2) called the *gravitational parameter*. There is a unique gravitational parameter associated with every primary body. [Ref. 2:pp. 14]

Another means of expressing the gravitational force associated with the primary body is called the constant of gravitation (k). The constant of gravitation is often left as k^2 to avoid the requirement of obtaining the square root. This value is also local to the system under study and is [Ref 4:pp. 23]:

$k = \sqrt{((G\ M_{earth}) \div a_{earth}^3)}$, where a_{earth} is the Earth 'a semi-major axis.

For elliptical orbits there are some values which are constant, regardless of the satellite's position. Specific mechanical energy (ε), specified as energy per unit mass, is constant throughout the orbit of the satellite. Since total energy consists of kinetic and potential terms, the satellite must "trade off" one form of energy for another as it changes position and distance. This relationship is given by [Ref. 2:pp. 16]:

$$\varepsilon = (v^2 \div 2) + (\mu \div r)$$

The angular momentum vector (**h**) is another orbital motion constant. Angular momentum is the cross product of the satellite's position (**r**) and velocity (**v**) vectors. Both of these vectors are measured with the center of the primary as their origin. The magnitude of the angular momentum vector is [Ref. 2:pp. 18]:

$$h = r\ v \cos \psi$$

Figure 2 shows how the flight-path angle (ψ) is defined as the angle measured from a line perpendicular to the position vector (called the *local horizontal*) to the velocity vector. The flight path angle will vary as the satellite orbits the Earth because **r** and **v** are also changing. One method of computing the magnitude of angular momentum is to select the distance and velocity at either perigee or apogee. The flight path angle at these positions is the same as the local horizontal ($\cos 0° = 1$) and:

$h = r_{perigee}\ v_{perigee} = r_{apogee}\ v_{apogee}$ [Ref. 2:pp. 18]

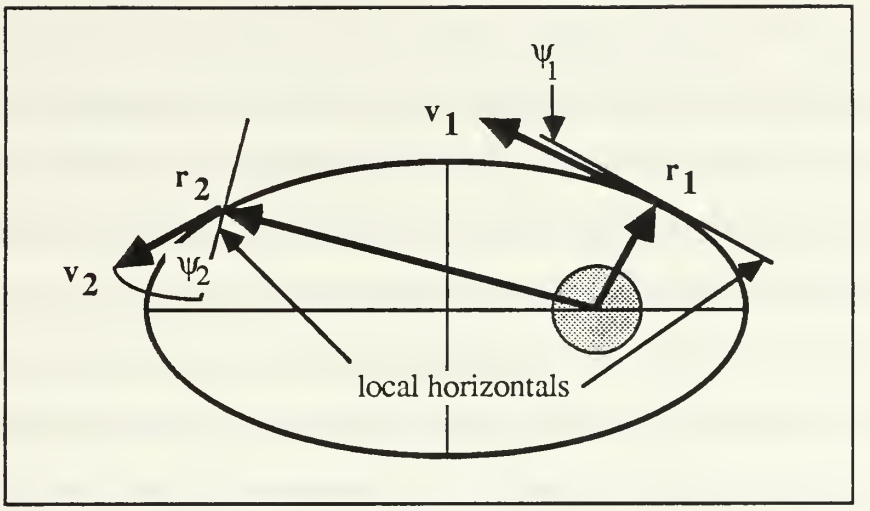


Figure 2. Relationship of Flight Path Angle to Position (**r**) and Velocity (**v**) Vectors at Two Selected Locations

3. Relationships between the Constants of the Motion and Geometry

By knowing some of the physical interpretations of the orbit and the geometric properties of an ellipse, it is possible to define several important new equations. Each of these relationships can, in turn, produce other ways of representing information about some special satellite orbits, such as circular orbits and escape velocities.

One important attribute of a satellite's orbit is the period (P) or time that the satellite takes to make one complete revolution around its primary. Earlier it was stated (in Kepler's Second Law) that the orbits of two satellites around the same primary were related by their periods and their mean distances from the primary. Because of this relationship it is possible to derive an equation that calculates the period of any satellite solely in terms of its semi-major axis (a) and the characteristics of the primary body (μ) it is orbiting. This equation is [Ref. 2:pp. 33]:

$$P = 2 \pi a^{1.5} \div \sqrt{\mu}$$

The addition of a definition for a new constant called the *mean motion* ($n = \sqrt{\mu} \div a^{1.5}$) allows a more compact representation of the period as [Ref 2:pp. 185]:

$$P = 2 \pi / n$$

A measurement that is related to period is the number of revolutions that the satellite makes around the Earth per day. The number of revolutions per day is simply obtained by dividing the length of the day by the period of the satellite. Unless the orbit is exactly circular, the satellite's speed or magnitude of velocity (v) is not constant as the satellite moves around the Earth. A satellite will move at a greater velocity the closer it is to the Earth and will move slower the further that it moves away from the Earth. It is still possible to calculate the satellite's speed (v) at any position (r) along the orbit (even if it is not a perfect circle) by using relationships similar to those used to derive a formula for the period. This relationship is called the *vis-viva* law or energy integral and is [Ref. 4:pp. 344]:

$$v^2 = \mu \left((2 \div r) - (1 \div a) \right)$$

The *escape velocity* is the amount of energy necessary to cause the satellite's kinetic energy to overcome the gravitational attraction of the primary. By allowing the size of the orbit to become infinite (i.e., $a \rightarrow \infty$) we can also obtain an equation for calculating a satellite's escape velocity. While this quantity is not directly related to the closed orbit problem, it does compute how much additional velocity ($v_{esc} - v$) would be required to launch a deep space probe from a given position (r) along the orbit. The equation for the total magnitude of velocity required to escape the Earth's gravitational field is [Ref. 2:pp. 35]:

$$V_{escape} = \sqrt{(\mu (2 \div r))}$$

4. Earth Coverage and Line of Sight Distance to the Horizon

The operating altitude of a satellite above the Earth's surface is an important consideration when choosing the satellite's orbit. Most satellites either observe the surface of the Earth directly or have signal strength and area coverage requirements for

communication with ground stations. More of the Earth's surface will be visible the higher a satellite is placed in orbit. Conversely, the level of detail or signal strength will decrease as the satellite is moved away from the Earth.

The farthest position on the surface of the Earth that is still visible to the satellite is called the *line of sight distance to the horizon.* The Earth's atmosphere may decrease or attenuate the signal below measurable levels before the satellite is below the horizon. Here, an angle above the horizon (γ) is specified above which the signal will remain strong enough for detection. The line of sight distance formula can be calculated by use of the Law of Sines and the relationships shown in Figure 3.

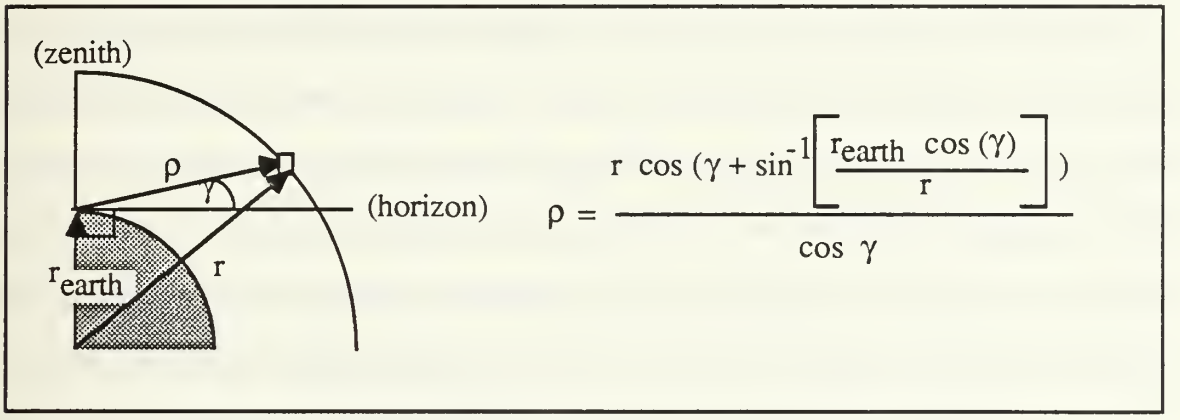


Figure 3. Line of Sight Distance (ρ) to the Horizon.

Another important orbital radius (r) selection criterion is the amount of the Earth's surface a satellite is capable of observing. The distance that a satellite can view from one horizon to another (normally taken at right angles to the motion of the satellite) is called the *geometric swath width* (GSW). This distance is an approximation because it assumes that the Earth is a smooth sphere and does not consider any sensor view angle restrictions that may exist. The equation for geometric swath width is [Ref. 5:pp. 40]:

$$GSW = 2\, r_{earth}\, \cos^{-1} (r_{earth} \div r)$$

A satellite in general is moving relative to a given location on the Earth's surface. The geometric swath width will form a path ("swath") of coverage over the Earth's surface that describes the satellite's useful coverage or field of view.

C. TIME

Throughout the discussion so far the measurement of specific values for a satellite has been taken at specific positions along the satellite's orbit. Therefore the notion of when the particular measurement was taken has not yet been introduced. Adding time as an additional coordinate element provides a tool for determining the satellite's position at a time other than when the measurement was taken.

Local time changes with an observer's longitude on the Earth's surface so there is a necessity for an absolute time system and an easy means to convert to and from this time scale. The most common units for measuring time are hours, minutes and seconds. However, because of the definition of a day as one rotation of the Earth, it is also possible to consider time as an angular measure of degrees or radians [Ref 4:pp. 235]. This relationship allows an easy conversion to and from location (longitude) and local time.

Unfortunately, the concept of time is also clouded slightly because a day is most commonly measured by the direction of the sun from an observer but a day is more correctly (for the purposes of orbits) measured by the position of the stars. The difference arises because as the Earth rotates counter-clockwise around its axis it is also orbiting around the sun The *solar* day is about 3 minutes 57 seconds shorter than the *sidereal* day, which is measured relative to the stars. [Ref. 2:pp.101]

1. Solar Time

The solar day begins at midnight, which is when the sun is exactly opposite its noon (directly overhead) position. However, the solar day is not constant because the tilt of the Earth's equatorial plane and elliptical shape of the Earth's orbit cause some slight

variations in the length of each day of the year. Therefore an average measure, called the mean solar day was established as being equivalent to 1.0027379093 days of sidereal time [Ref. 2:pp. 101]. Two commonly used time systems are defined by using different reference points and mean solar time: Standard time and Universal time.

Standard (or local) time is the time that is measured by an observer at the observer's longitude on the Earth's surface. The Earth is divided into 24 time zones, spaced 15° apart, that are each equivalent to one hour of solar time. Local time zones are the most common form of time keeping and keeps the hour of the day roughly matching the position of the sun (at a specific latitude) over the Earth's surface.

The concept of Universal (or Zulu or Greenwich mean) time is the measuring of time from one specific meridian (the Greenwich meridian, defined to be 0° Longitude), regardless of an observer's location. This allows a common system of units for communicating the measurement of time without having to consider the effect of local time zones. To convert from Local time to Universal time (UT) all that must be done is add to the Local time the number of time zones (hours) that the local meridian is to the west of the Greenwich meridian. For example, at 1:18 P.M. in Monterey, California (approximately 122° West = 122° ÷ 15° = 8 whole time zones), the time is 1:18 + 8 hrs or 9:18 P.M. Universal time. Time is normally stated in standard time (i.e., without daylight savings time) and by 24 hour clock, so 9:18 PM UT is expressed as 21:18 hours UT. [Ref 2:pp. 103]

2. Sidereal Time

Sidereal time is the time reference used for calculating the orbit of a satellite because this time system is the natural system for determining the motion of the satellite. The sidereal day, by definition, begins when the Greenwich meridian is aligned with a fixed direction called the *vernal equinox*. Vernal Equinox is the point where the sun

crosses the celestial equator from south to north [Ref. 4:pp. 8] To convert from solar time to sidereal time, the solar time must first be converted into Universal time and then adjusted to determine sidereal time. This adjustment is a physical characteristic of the Earth and is measured by observatories and compiled into the *American Ephemeris and Nautical Almanac* [Ref 4:pp. 235].

Once again there exists the concepts of universal (Greenwich Sidereal) and position dependent (Local Sidereal) time. The Greenwich Sidereal time (θ_g) is the value that is measured and tabulated in almanacs for use in converting from solar to sidereal time systems. Because the measurement of sidereal time is based on the position of the stars from a specific location on the Earth's surface, Local Sidereal time is often expressed as an angular measure (λ_E), called the *Local Hour Angle* (LHA). Local sidereal time may be computed by:

$$\theta = \theta_g + \lambda_E$$

If Greenwich Sidereal time is not known, but Universal Time (solar time) is available, then a good approximation for local sidereal time is still obtainable. This is accomplished by calculating the time difference between a known value (θ_{g0}, found in the *Nautical Almanac*) and the current time according to the following formula:

$\theta = \theta_{g0} + \rho' (t - t_0) + \lambda_E$, where ρ' is the angular rotation rate of the Earth, $t - t_0$ is the time difference from the specified time (t) to a known Greenwich Sidereal time (t_0). The values for θ_{g0} are generally compiled when t_0 is 0000 hrs UT. [Ref. 2:pp. 100]

D. CANONICAL UNITS

A topic that is closely related to the physical characteristics of a satellite and its primary is the use of system specific measurement units called characteristic or *canonical units*. Canonical means that the fundamental measuring unit for everything in the orbit system is based on the primary's radius and a special orbit called the reference orbit.

This conversion from more commonly used units, such as the metric system, to canonical units is really just a mathematical convenience and does not change the basic relationships in any way. Canonical units simplify many of the basic formulas and are used by many complex orbital models to increase computational efficiency. The standard library is not optimized for speed but is intended to demonstrate the fundamental concepts behind the computation of orbits. Therefore all input and output is specified in more familiar metric units. Additionally, routines for conversion between canonical and metric units is provided for general use in the library.

Distance units are based on the mean equatorial radius of the Earth (r_{earth} = 6378.145 km); thus a satellite at an altitude of 750 km is at $(750 + r_{earth})/ r_{earth}$ = 1.117589 Earth radii or distance units from the center of the Earth. Time units are specified by using the speed of a hypothetical circular orbit that is located at the Earth's surface ($r = r_{earth}$). If the speed of the orbit is defined such that the satellite's period is exactly 2π, then the time unit (τ, called a *herg*) is equivalent to $\tau = r_{earth}^{1.5} \div \sqrt{\mu}$ = 806.8118744 seconds. As a consequence of the above relationships, the canonical speed unit (distance ÷ time) is 7.90536828 km/sec in metric units. [Ref 2:pp. 40-43]

E. CORRECTION FOR ASPHERICAL GRAVITATIONAL POTENTIAL

As was initially stated, the two body closed orbit problem simplified the modeling of orbits at the expense of a more exact representation of the real world. There are some significant aspects of orbital motion that this simple model cannot describe. These deviations are often called *perturbations*, because they cause the satellite to follow a path that is different from the one expected by the simple two body model. [Ref. 2:pp. 385]

Some examples of perturbations are radiation pressure from the sun, changes in gravitational influence, and atmospheric drag. One possible form of a changing gravitational influence is the presence of other astronomical bodies. Correction for this

perturbation is not done by the standard library because of the small magnitude of changes involved (especially for low earth orbits) and the complications that would have to be added to the introductory astrodynamics presented in the standard library. To properly account for the force of atmospheric drag, the model must calculate the density of the atmosphere at high altitudes. The standard library and most predictive models of satellite motion avoid the difficulty in modeling the upper reaches of the atmosphere by ignoring the lowest orbital altitude regime and only model satellite orbits at altitudes greater than 850 kilometers. [Ref. 6:pp. 87]

Unless the satellite is in an extremely low orbit and experiencing atmospheric drag, the largest perturbative force results from the Earth being slightly more massive around the equator. Because the mass of the Earth is distributed non-uniformly, there are slight deviations in the gravitational attraction of the Earth at different locations. This *aspherical gravitational potential* results in two distinct changes to a satellites orbit: a gradual rotation of the orbital plane (regression) and a rotation in the orientation of the ellipse [Ref. 2:pp. 156-159]. These effects are discernible over several orbits, therefore it is important that the library provide a mechanism to calculate this effect. The mass distribution of the Earth is irregular in every direction, but is most affected by latitude. The model commonly chosen to represent the Earth is axially symmetric with different variations in mass distributions for each hemisphere. The model's mathematical foundation is an infinite series representing the Earth's gravitational potential [Ref. 4:pp. 174]:

$U = (k^2 M_{earth} \div r) [1 - \Sigma_{k=2} (J_k^{(0)} \div r^k) P_k \sin \delta]$ where $J_2^{(0)}$, $J_3^{(0)}$, ..., are the coefficients of the Earth's gravitational potential called the *zonal harmonics*, and P_2, P_3, ..., are Legendre polynomials.

Although this equation is not used directly in the standard model, its formulation is presented because this representation is the basis for the development of the equations for two body perturbed motion.

IV. CELESTIAL COORDINATE SYSTEMS AND TRANSFORMATIONS

There are many possible coordinate systems that can be used to describe the position and motion of an earth orbiting satellite. All coordinate systems used by this library are orthogonal, that is each of the three unit vectors are mutually perpendicular, and additionally the unit vectors also follow the "right hand rule" convention in their ordering. The three most common fundamental planes for these systems are the orbit itself, the Earth's equatorial plane and an observer's (on the Earth's surface) horizon. The fundamental plane always establishes one vector normal to the plane but there is still much freedom of choice for the two remaining in-plane vectors. The geocentric-equatorial, right ascension-declination, geographic, topocentric, and perifocal coordinate systems are directly supported by the standard library. Many of the other possible coordinate systems can be derived easily from one of these five if they are required.

An important coordinate system property is whether the coordinate system is inertial, quasi-inertial, or not inertial at all. Inertial systems can be defined as having axes that are not in accelerated or rotational motion [Ref. 2:pp. 9]. The study of mechanics is greatly simplified in inertial or quasi-inertial (approximations to inertial) coordinate systems because the terms resulting from rotational motion do not have to be considered.

A. GEOCENTRIC-EQUATORIAL (IJK) COORDINATE SYSTEM

The geocentric-equatorial coordinate system is probably the most obvious one; it is simply the two dimensional rectangular coordinate system extended into three dimensions. Figure 4 shows how this coordinate system uses the equator of the Earth as its fundamental plane (with the z-axis normal to the plane and pointing towards the celestial

north pole) and the x-axis points in the direction of vernal equinox. The unit vectors **I**, **J**, **K** are often used as the shorthand notation **IJK** for referring to a satellite's position (x,y,z) in the geocentric-equatorial coordinate system. [Ref. 2:pp. 54-57]

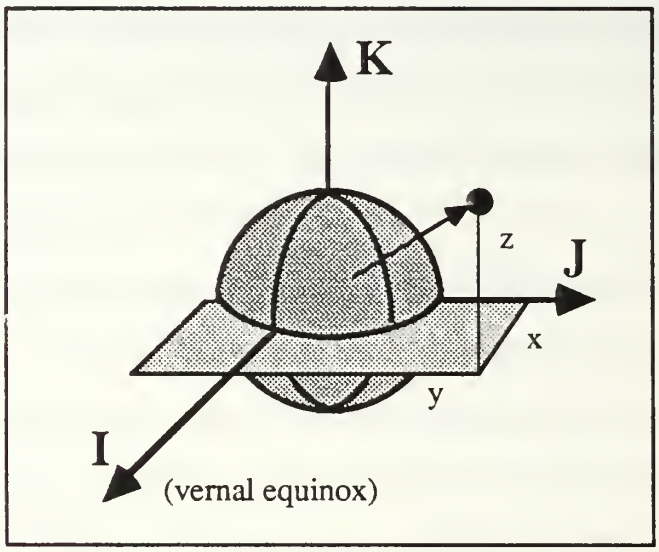


Figure 4. Geocentric-Equatorial Coordinate System

B. RIGHT ASCENSION-DECLINATION (R.A.-DECL) COORDINATE SYSTEM

The right ascension-declination coordinate system is most commonly used by astronomers to catalog the positions of stars and other heavenly bodies. In this system the position of an object is obtained by projecting the pointing vector for the object against a sphere of infinite radius (the celestial sphere) and determining two angles called right ascension (α) and declination (δ). These angles are measured positive upward from the equatorial plane and counter-clockwise positive from the vernal equinox. The origin of the coordinate system can be chosen as the Earth's center, the observer's position, or any other arbitrary point because the celestial sphere is infinitely large Ref. 2:pp. 56-57]. The distance from the object to the chosen origin is called the radial distance (r) and is an essential element when translating into other coordinate systems using different origins.

The right ascension-declination coordinate system shares a common principle pointing direction (i.e., in the direction of vernal equinox) with the geocentric-equatorial coordinate system as is shown in Figure 5.

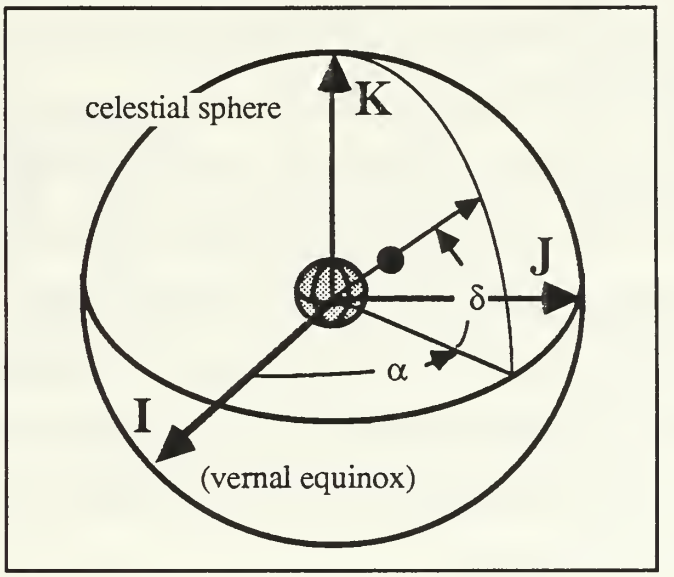


Figure 5. Right Ascension-Declination Coordinate System

C. GEOGRAPHIC (φλh) COORDINATE SYSTEM

The geographic coordinate system uses the equator of the Earth as its fundamental plane and the Greenwich (Prime) meridian as the direction of its principle pointing vector. This is not an inertial system as the Greenwich meridian rotates through 360 degrees every solar day [Ref. 4:pp. 96-97]. The coordinates in the geographic system are longitude (east or west), latitude (geocentric or geodetic), and altitude above the Earth's surface. It is a matter of preference whether longitude is specified between -180 and 180 degrees when measuring longitude from east to west (such as is used in the *American Ephemeris and Nautical Almanac* [Ref. 4:pp. 97]) or if an east/west longitude convention is used exclusively and longitude is allowed to assume values from 0 to 360 degrees.

The standard library adopts the common convention of using east longitude (λ_E) exclusively. Thus all longitudes are measured counterclockwise from the normal vector of the fundamental plane (i.e., the equatorial plane).

Latitude measurements also suffer from complications because the Earth is not a perfect sphere. Geocentric latitude (ϕ_g) is the angle measured from the equatorial plane to the position vector of the object being referenced (generally an observer's position on the Earth's surface) [Ref. 4:pp. 97]. This angle can be considered to be referenced from a perfectly spherical Earth and is the latitude most easily obtained from spherical trigonometry. Geodetic latitude (ϕ) is measured from the equatorial plane to the normal vector at the observers position on a "reference ellipsoid". The reference ellipsoid is an approximate model of the Earth's oblateness based on an ellipse rotated around the Earth's axis. Geodetic latitude is the commonly referred to latitude found on most maps and charts of the world. [Ref. 2:pp. 94]

Astronomical latitude, another type of latitude sometimes used, is an angle measured between the direction of the local gravitational field and the equator. Since astronomical latitude deviates only slightly from the reference ellipsoid [Ref. 2:pp. 94] it is not used by the standard library.

Because altitude is the object's distance above the reference ellipsoid, the altitude specified in this coordinate system is dependent on which form of latitude is being used. If geocentric latitude (ϕ_g) is specified then the altitude (h_g) is measured above a reference sphere of constant radius (Re = 6371.086 km) [Ref. 6:pp. 24]. For geodetic latitude (ϕ) the altitude (h) is the difference between the object's radial position and the ellipsoid radius at the specified latitude (see Figure 6). To further refine altitude by taking the irregularity of the Earth's surface into account (e.g., mountains and valleys) either the observer must have knowledge of the Earth's topography at that latitude and longitude or a complicated model of the Earth's surface must be developed. The standard model simply provides

altitude measured from the standard ellipsoid (or sphere); it is the responsibility of the using program to further refine that value.

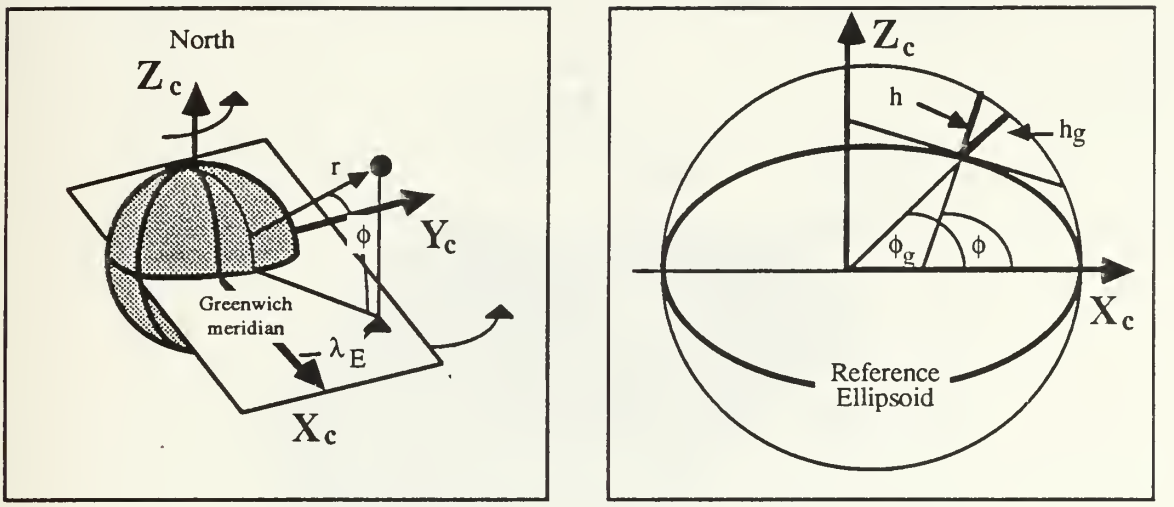


Figure 6. Geographic Coordinate System

D. TOPOCENTRIC (SEZ) COORDINATE SYSTEM

Another coordinate system that is inherently non-inertial is the topocentric coordinate system. The topocentric coordinate system uses the observer's position as its origin, and the observer's horizon as its fundamental plane. Figure 7 shows the $\mathbf{Z_h}$ vector as the normal vector pointing upwards from the surface of the Earth and the two in-plane vectors ($\mathbf{X_h}$, $\mathbf{Y_h}$) pointing positive towards the south and east, respectively. The coordinate axes, **SEZ** (south, east and up), are generally used as shorthand notation for the topocentric coordinate system. [Ref. 2:pp. 84]

This coordinate system is extremely useful because most methods of determining satellite orbits rely on radar or visual sightings from observer stations that are on the Earth's surface. Position information is provided as range (ρ) and two angles which specify the direction from which the range information is obtained. These angles are azimuth (A_Z, measured clockwise from -S) and elevation (E_l, measured from the

horizontal plane). Velocity can be specified as the time rate of change of each of the position elements (ρ', A_Z', and E_l').

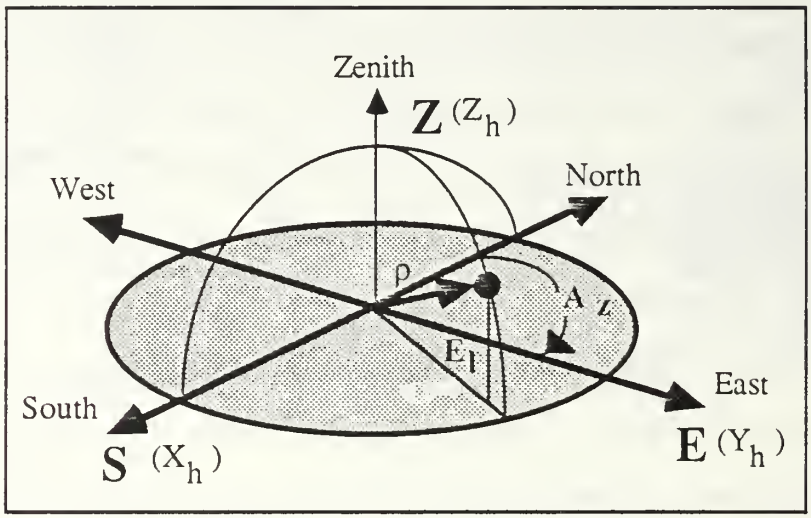


Figure 7. Topocentric Coordinate System

E. PERIFOCAL (PQW) COORDINATE SYSTEM

The perifocal coordinate system is perhaps the most familiar coordinate system to the introductory student of astrodynamics. The fundamental plane is the orbit itself, with the principle direction (**P**) pointing towards the point of the satellite's closest approach to the Earth (periapsis). The center of the Earth is the perifocal coordinate system's origin and is also one foci of the ellipse. The other in-plane vector (**Q**) is established at a right angle (in the direction of orbital motion) to the principle vector. The remaining vector is chosen so that these vectors will form a right-handed orthogonal coordinate system, Figure 8. Because of the commonly used designation for the perifocal coordinate axes, this system of coordinates is often referred to as the **PQW** coordinate system. The position and velocity elements in this coordinate system are denoted by (x_ω, y_ω, z_ω) and (x_ω', y_ω', z_ω'), respectively. This coordinate system allows the two-body motion of a satellite to remain within the fundamental plane (the **W** components are exactly zero). This is

valuable for simplifying the amount of calculations necessary for the orbit prediction problem. It is worth noting that periapsis is not defined for a circular orbit (e = 0) and complicates the use of this coordinate system for orbits with low eccentricities. [Ref 4:pp. 115]

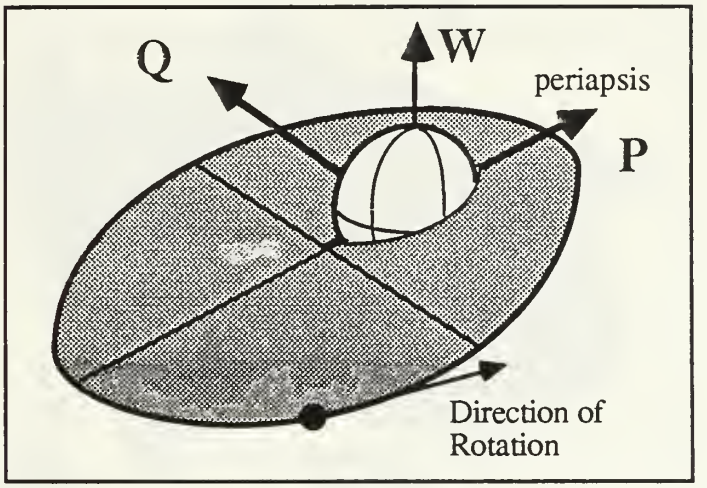


Figure 8. Perifocal Coordinate System

F. KEPLERIAN ELEMENTS

A set of orbital elements with great historical interest are the Keplerian or Classical Elements. These elements are often used to define the relationship between the orbital and the equatorial planes and are defined as the semi-major axis (a), eccentricity (e), inclination (i), longitude of the ascending node (Ω_0), argument of perigee (ω_0), and time of periapsis passage (T) [Ref. 2:pp. 58]. The concepts of semi-major axis, eccentricity and inclination have been already introduced, and with the use of two new angles (longitude of the ascending node and argument of perigee) will define the rotation of the orbital plane from the equatorial plane. The longitude of the ascending node is the angle, measured counter-clockwise, from vernal equinox to the orbit's northward crossing of the equator. The vector from the center of the Earth to this point is called the line of nodes. The

argument of perigee, an angle measured counter-clockwise from the line of nodes to periapsis, is the final angle necessary to determine the orbital plane. These relationships are illustrated in Figure 9.

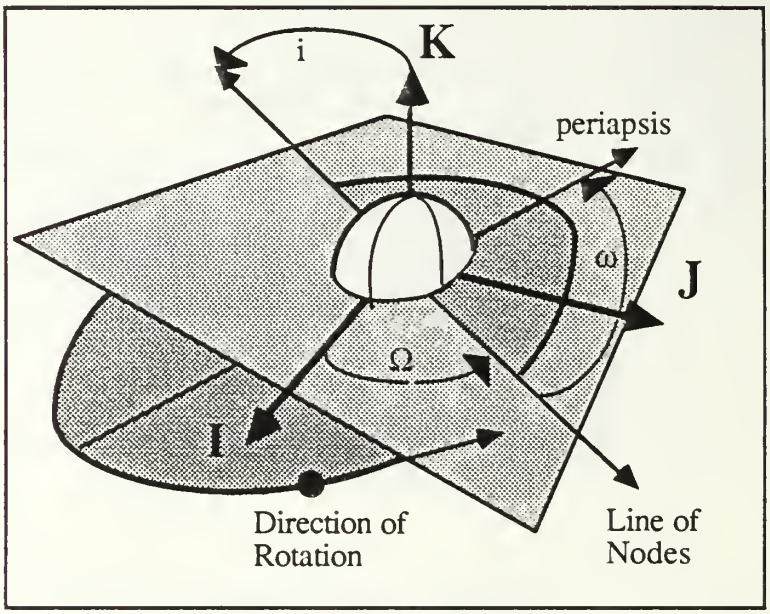


Figure 9. Keplerian Elements

The sixth element (time of periapsis passage) determines the location of the satellite along the orbit. The reason that subscripts exist on the symbols for argument of perigee and longitude of the ascending node is because they are more changeable with respect to time than the remaining elements. The other five Keplerian elements (or any other orbital element set) are defined at a time called epoch. The time of periapsis passage can be expressed as the instant in time that the satellite is at perigee [Ref. 2:pp. 60]. If the epoch time does not correspond with the time of periapsis passage, then two separate values (anomaly and arbitrary epoch time-t_0) would be required to define the time element [Ref. 6:pp. 58]. Anomaly is an angle measured in one of three standard ways that defines the location of the orbiting object from the perigee position.

There are three common measurements of anomaly called the *true anomaly*, *eccentric anomaly* and *mean anomaly*. The true anomaly (ν_0) has a direct geometric representation and is simply the polar angle that is measured from perigee to the object's position. The eccentric anomaly is determined by projecting the orbital path of the satellite against an *auxiliary circle* with a radius equal to the semi-major axis of the orbit. This relationship is illustrated in Figure 10. For circular orbits the auxiliary circle is the real orbital path and the eccentric anomaly is identical to the true anomaly.

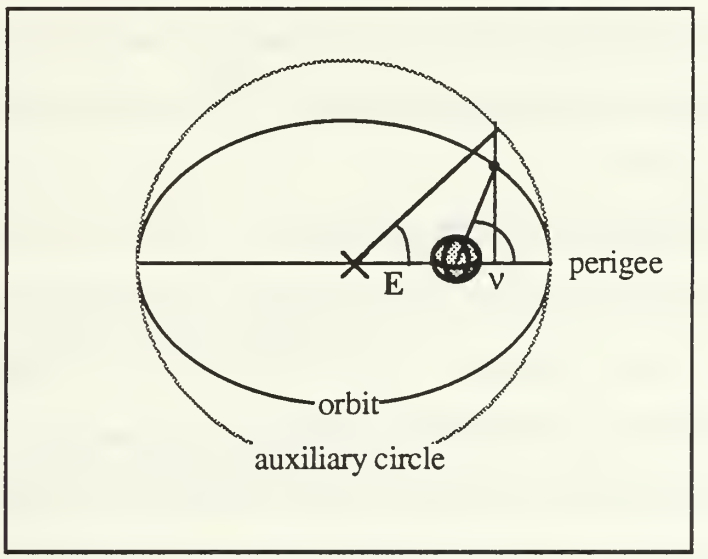


Figure 10. True Anomaly (ν) and Eccentric Anomaly (E) Relationship

A satellite does not travel at uniform speed in an elliptical orbit, therefore the rate of change of the true anomaly will also vary. The mean anomaly (M_0) measures the object's position from perigee along a hypothetical orbit of uniform average speed (called the mean motion constant; n). Mean anomaly has no geometric interpretation but is defined in general (for any arbitrary time, t) by the equation:

$$M = n\,(t - T) = M_o + n\,(t - t_0) = E - e \sin E$$

The last equivalence is the common form of expressing Kepler's Equation or the so-called *Kepler problem*. Because this equation relates position (E) and time (M_0, t_0, and t) it provides a formula for predicting the position of a satellite at any time. This formula requires finding a solution for eccentric anomaly by using these other (known) values. Since E is expressed by a transcendental function and a polynomial of degree one, a closed form solution to this equation does not exist. The search for an efficient method to solve this equation is the reason that this formula has come to be known as the Kepler problem [Ref. 2:pp. 220]. The analytical and numerical solutions to the Kepler problem are useful for predicting the position of a satellite at a time other than epoch. and several solutions will be discussed in the next chapter.

G. TRANSFORMATION FORMULAS

Several formulas for transforming position from one coordinate system to another are provided based on the coordinate systems defined in this chapter. The first few transformations make use of Keplerian elements as well as system coordinates, but the remaining transformations are defined solely by individual coordinate systems. If an explicit transformation is not listed, then it is generally possible to obtain the desired coordinate system via one or more intermediate transformations. As a source of additional transformations, an appendix to Escobal provides a compilation of no less than thirty-six basic coordinate transformations [Ref. 7:pp. 393-422].

1. Keplerian to PQW

Using true anomaly (ν) or eccentric anomaly (E) the three **PQW** position elements are [Ref 2:pp.45-46]:

$$x_\omega = r \cos \nu = a (\cos E - e)$$
$$y_\omega = r \sin \nu = a (\sin E \sqrt{(1 - e^2)})$$
$$z_\omega = 0$$

2. PQW and Keplerian to IJK

Using argument of perigee (ω_0), longitude of the ascending node (Ω_0) and inclination (i) the three **IJK** position elements are [Ref 6:pp. 62-64]:

Use transformation 1 to obtain x_ω and y_ω.

$$x = x_\omega (\cos \omega_0 \cos \Omega_0 - \sin \omega_0 \sin \Omega_0 \cos i) + y_\omega (-\sin \omega_0 \cos \Omega_0 - \cos \omega_0 \sin \Omega_0 \cos i)$$

$$y = x_\omega (\cos \omega_0 \sin \Omega_0 + \sin \omega_0 \cos \Omega_0 \cos i) + y_\omega (-\sin \omega_0 \sin \Omega_0 + \cos \omega_0 \cos \Omega_0 \cos i)$$

$$z = x_\omega (\sin \omega_0 \sin i) + y_\omega (\cos \omega_0 \sin i)$$

3. IJK and Keplerian to PQW (inverse of transformation 2)

Using argument of perigee (ω_0), longitude of the ascending node (Ω_0) and inclination (i) the three **PQW** position elements are [Ref 7:pp. 417-418]:

$$x_\omega = x (\cos \omega_0 \cos \Omega_0 - \sin \omega_0 \sin \Omega_0 \cos i) + y (\cos \omega_0 \sin \Omega_0 + \sin \omega_0 \cos \Omega_0 \cos i + z (\sin \omega_0 \sin i)$$

$$y_\omega = x (- \sin \omega_0 \cos \Omega_0 - \cos \omega_0 \sin \Omega_0 \cos i) + y (- \sin \omega_0 \sin \Omega_0 + \cos \omega_0 \cos \Omega_0 \cos i) + z (\cos \omega_0 \sin i)$$

$$z_\omega = x (\sin \Omega_0 \sin i) + y (- \cos \Omega_0 \sin i)) + z (\cos i)$$

4. SEZ to IJK

Using geodetic latitude (ϕ_g), the Earth's equatorial radius (a_e = 6378.145 km), the Earth's polar radius (b_e = 6356.785 km) and the reference ellipsoid's eccentricity (e_e = 0.08182) the three **IJK** position elements are [Ref 2:pp. 85-101]:

The topocentric position vector ρ is obtained first by $\rho = \rho_S \mathbf{S} + \rho_E \mathbf{E} + \rho_Z \mathbf{Z}$ where:

$\rho_S = - \rho \cos E_l \cos A_z$
$\rho_E = \rho \cos E_l \sin A_z$
$\rho_Z = \rho \sin E_l$

Now, by making use of geodetic latitude (ϕ_g) and defining local sidereal time (θ) as $\theta = \theta_g + \lambda_E$ (θ_g is the angle between **I** and the Greenwich meridian at epoch and λ_E is the east longitude of the observer) we find:

$x = \rho_S (\sin \phi_g \cos \theta) + \rho_E (- \sin \theta) + \rho_Z (\cos \phi_g \cos \theta)$
$y = \rho_S (\sin \phi_g \sin \theta) + \rho_E (\cos \theta) + \rho_Z (\cos \phi_g \sin \theta)$
$z = \rho_S (- \cos \phi_g) + \rho_Z (\sin \phi_g)$

The observer position (or station coordinates) in **IJK** coordinates is:

$x_r = | (a_e / (1 - e_e^2 \sin^2 \phi_g)^{1/2}) + h | \cos \phi_g$
$z_r = | ((a_e (1 - e_e^2)) / (1 - e_e^2 \sin^2 \phi_g)^{1/2}) + h | \sin \phi_g$

$x_S = x_r (\cos \theta)$
$y_S = x_r (\sin \theta)$
$z_S = z_r$

5. Right Ascension-Declination to IJK

Using spherical trigonometry, the three **IJK** position elements can be obtained by [Ref 4:pp. 105]:

$x = r \cos \delta \cos \alpha$
$y = r \cos \delta \sin \alpha$
$z = r \sin \delta$

6. IJK to Right Ascension-Declination (inverse of transformation 5)

Using spherical trigonometry, the three **Right Ascension-Declination** position elements can be obtained by [Ref 7:pp. 397-398]:

$r = (x^2 + y^2 + z^2)^{1/2}$
$\alpha = \tan^{-1}(y/x)$ where $0° \leq \alpha \leq 360°$
$\delta = \tan^{-1}(z / (x^2+y^2)^{1/2})$ where $-90° \leq \delta \leq 90°$

7. IJK to Geographic ($\phi \lambda h$)

By computing the displacement effect of the Earth's rotation, the three **Geographic** position elements can be derived from [Ref 6:pp. 77-78]:

$\rho = \rho_0 + \rho' (t - t_0)$ where t_0 is the time when the Greenwich meridian is aligned with the **I** axis ("sidereal epoch") and $\rho'=7.292115856 \times 10^{-5}$ rad/sec is the Earth's rotation rate. ($o° \leq \rho \leq 360°$)

$x_{earth} = x (\cos \rho) + y (\sin \rho)$
$y_{earth} = x (- \sin \rho) + y (\cos \rho)$
$z_{earth} = z$

$r^2 = x_{earth}^2 + y_{earth}^2 + z_{earth}^2$

$\phi = \sin^{-1} (z_{earth} / r)$
$\lambda = \tan^{-1} (y_{earth} / x_{earth})$
$h = r - r_{earth}$, where r_{earth} is the mean radius of the Earth

To convert from geocentric latitude/altitude to geodetic latitude/altitude [Ref. 6:pp. 27]:
$\phi_g = \tan^{-1} [\tan \phi \div (1-e_{earth}^2)]$
$h_g = r - r_{earth}(\phi_g)$, where r_{earth} is the radius of the ellipsoid representing the Earth as a function of latitude.

8. Geographic to IJK (inverse of transformation 7)

By computing the displacement effect of the Earth's rotation, the three IJK position elements can be derived from [Ref 7:pp. 399-400]:

If the geographic coordinates are given with respect to geodetic latitude (ϕ_g, h_g,λ) they must be converted into geocentric latitude by [Ref 6:pp.27]:
$\phi = \tan^{-1} [\tan \phi_g (1-e_{earth}^2)]$, $-90° \le \phi \le 90°$

$r_{earth}^2 = (a_{earth}^2 [1- (2f - f^2)]) / (1- (2f - f^2) \cos^2 \phi_g)$
$r = (r_{earth}^2 + h_g^2 + 2 r_{earth} h_g \cos (\phi - \phi_g)$, where r_{earth} is the oblate Earth radius at latitude ϕ_g, and f is the flattening of the Earth ($f = 1 - \sqrt{(1-e_{earth}^2)}$).

$\theta = \theta_g + \theta'(t - t_0) - (360° - \lambda_E)$, $0° \le \lambda \le 360°$
$\Delta\phi_g = \sin^{-1} ((h_g/r) \sin (\phi - \phi_g))$, where $\Delta\phi_g$ is the difference between geocentric latitude and declination.

$\delta = \phi_g + \Delta\phi_g$

$x = r (\cos \delta \cos \theta)$
$y = r (\cos \delta \sin \theta)$
$z = r (\sin \delta)$

V. ORBIT PREDICTION

A. STATEMENT OF THE PROBLEM

The problem of determining the position of a satellite at an arbitrary time (t) from an initial set of coordinates (measured at epoch, T) has been studied since the work of Kepler [Ref. 2:pp.212-222]. The classical solution is tied closely to a traditional set of coordinates: the so called *Classic* or *Keplerian Elements*. Consequently, the procedure used to obtain the satellite's time dependent position (as outlined below) is stated using Keplerian Elements. To simplify calculation of the new position, it will initially be defined in the perifocal (**PQW**) coordinate system. From the **PQW** coordinate system a simple series of transformations can be made to obtain the satellite's position in whatever coordinate system is desired. The orbit prediction problem may be summarized as:

Given: The satellite position elements a, e, i, Ω_0, ω_0, M_0 at epoch (t_0).

Find: The position of the satellite at an arbitrary time (t).

B. OUTLINE OF THE SOLUTION

There are four distinct steps required to obtain a new position for the satellite. These steps are:

1. Compute the Mean Anomaly (M) at time t
2. Determine the Eccentric Anomaly from the Mean Anomaly (i.e., use a solution to the Kepler Problem)
3. Obtain the position vector in the Perifocal (**PQW**) Coordinate System
4. Transform the result into the coordinate system of choice (see section on orbital transformations)

1. Compute the Mean Anomaly (M) at Time t

The shape of the orbit would remain constant if the satellite was a massless point and subject only to gravity from a point source. If this were the case, then by knowing the change in time from the first position, it is possible (with some difficulty) to locate the satellite along this perfect ellipse. This simplification is called the two body problem and can often result in a sufficiently accurate computation of the satellite's position. The time dependent nature of this relationship is represented by the time that the satellite most recently passed perigee and the mean anomaly. The mean anomaly (M) may be represented as [Ref. 2:pp. 185]:

$M = n\,(t - T)$, where $T = t_0 - (M_0 \div n)$ is the *time of perifocal passage* and $n = \sqrt{(\mu \div a^3)}$ is the *mean motion constant.*

The most significant perturbation to add to the basic two body problem model is the aspherical gravitational potential of the Earth. The Earth's gravity is not constant because the planet's mass is not distributed uniformly. Thus, the force of gravitational attraction will vary periodically as the satellite makes a single revolution around the Earth. This type of perturbation is called *periodic* because the satellite deviates from its predicted orbit, yet returns to nearly the same location that it started from. Periodic perturbations typically cause no loss of the total energy for the satellite and do not cause much change to the values of a, e, and i. The values for Ω, ω, and M, however, are more easily subject to change by this type of perturbing force. [Ref. 6:pp. 87-89]

In earlier discussions, the analytic solution for the aspherical nature of the Earth's gravitational potential was expressed as an infinite series. Changes to each orbital element are calculated by multiplying their rate of change by the time difference from initial measurement $(t - t_0)$.

New values of the mean anomaly, longitude of the ascending node and argument of perigee are found by [Ref 6:pp. 94]:

$M = M_0 + M'(t - t_0)$

$\Omega = \Omega_0 + \Omega'(t - t_0)$

$\omega = \omega_0 + \omega'(t - t_0)$, where t_0 is an arbitrary epoch time and t is the instant in time under consideration. For the simple two body problem $M' = n$ and $\Omega' = \omega' = 0$.

The time rate of change for each of these variables is derived from a Taylor series expansion that uses as many terms as needed to obtain the desired accuracy and truncates the remaining terms. The first approximation to the change in M, Ω, and ω (due to the Earth's aspherical gravitational potential) is provided in Figure 11 [Ref. 6:pp. 94 - 95].

$$M' = n' = n\left[1 + \frac{3}{2} J_2 \frac{\sqrt{1 - e^2}}{p^2} \left(1 - \frac{3}{2} \sin^2 i\right)\right]$$

$$\Omega' = -\left(\frac{3}{2} \frac{J_2}{p^2} \cos i\right) n'$$

$$\omega' = -\left(\frac{3}{2} \frac{J_2}{p^2} \left(2 - \frac{5}{2} \sin^2 i\right)\right) n'$$

Figure 11. First Order Perturbative Effects

These small changes are a function of the orbital inclination, J_2 the 2nd harmonic coefficient (an emperically obtained constant), and p is the semi-parameter of an ellipse $p = a(1 - e^2)$. [Ref. 6:pp. 50]

The expansion to higher than first order effects is computational much more difficult than this first approximation. This complexity results from the increased number of terms that result from the expansion to the fourth harmonic coefficient. The use of second or higher order expansions will increase accuracy, but the effects of the aspherical gravitational potential may be appreciated by simply using first order terms.

2. Determine the Eccentric Anomaly from the Mean Anomaly

Because the solution to the Kepler problem ($M = E - e \sin E$) is transcendental, an iterative solution based on the Newton-Raphson method of root finding is used. The root in question is a solution to the equation: $M - E + e \sin E = 0$. This algorithm takes the form of [Ref. 2:pp. 222]:

(1) $M_n = E_n - e \sin E_n$

(2) $E_{n+1} = E_n + (M - M_n) \div (1 - e \cos E_n)$, where this equation is applied initially to $E_0 = M$ and then reapplied until the difference between M and M_n becomes small enough to be ignored.

(3) If true anomaly (ν) is also desired, it may be calculated from [Ref. 2:pp. 187]:
$\nu = \cos^{-1} [(e - \cos E) \div (e \cos E - 1)]$

3. Obtain the Position Vector in the Perifocal Coordinate System

The perifocal coordinate system (**PQW**) uses the orbit as its fundamental plane and therefore requires only two coordinates to fully specify the satellite's position. The z_ω coordinate is by definition always equal to zero. Based on coordinate transformation 1 and the computed values of M, Ω, and ω at time t, the position of the satellite can be calculated as:

$$x_\omega = r \cos \nu = a (\cos E - e)$$
$$y_\omega = r \sin \nu = a (\sin E \sqrt{(1 - e^2)})$$
$$z_\omega = 0$$

4. Transform the Result into the Coordinate System of Choice

Because the location of a satellite might be desired in coordinates other than the PQW, the section that discusses coordinate system provides methods to transform PQW coordinates into other common coordinate systems. It is exactly by this method that the simulation, using the standard library, determines new position vectors for every coordinate system other than PQW.

VI. COMMON ORBIT CLASSIFICATIONS

With minimal reference to the physical laws governing the motion of satellites, it is still easy to describe many of the fundamental characteristics of orbits. The most obvious attribute of a satellite is that it is constantly in motion. This motion is responsible for the satellite's track in the night sky and for providing the capability to match the rotation of the Earth. This second relationship forms the principle for global communications from *geostationary* satellites that appear to "hang" in one position over the Earth's equator.

The question then arises concerning the determination of relative position, area coverage or Earth locations visited for each class of satellite orbits. It is possible to create a partial, qualitative list of orbits that exhibit various interesting phenomena. This list is provided without an excessively detailed explanation of the parameters (found in earlier chapters) which are required to fully describe each orbit type.

A. ORBIT CLASSIFICATION BY ALTITUDE

1. Low Earth Orbit (LEO)

The lowest practical orbital altitude is normally defined to be 185 km. Orbits below this altitude decay rapidly due to atmospheric drag. This drag results in short orbital lifetimes, many of which are only a few days in duration [Ref. 2:pp. 152]. Above this altitude, the effects of atmospheric decay diminish rapidly, although some decay is still experienced by any satellite in LEO. The definition of an upper limit on altitude for low earth orbiting satellites is less specific than the lower one, but generally occurs around 1,000 km [Ref. 2:pp. 153]. These orbits are characterized as being nearly circular because there is little variation possible between the maximum apogee and minimum perigee.

LEO satellites exhibit short periods (i.e., from 87 to 105 minutes) that allow satellites to circle the Earth many times per day. Because the satellite is close to the Earth this variety of orbit is often used for making observation of the Earth's surface. By virtue of its closeness to the Earth's surface a satellite requires less energy to reach LEO than any other orbit. Therefore it is often used by heavier satellites, such as manned systems, and as a *parking* orbit for satellites that are destined for higher orbits.

2. High Earth Orbit (HEO)

A satellite in high earth orbit is one that spends the most of its time in altitudes from 5,000 to 19,300 km. A satellite is often placed in HEO to obtain a wider field of view of the Earth's surface or to eliminate the residual effects of atmospheric drag. Satellites operating at these altitudes revolve around the Earth at a slower rate than a LEO satellite, but in general will circle the Earth at least once per day.

B. ORBIT CLASSIFICATION BY INCLINATION

The inclination of a satellite is the angle that the plane in which the satellite is moving is tilted from the Earth's equator. Inclination can vary from 0 to 180°, where 0° is an *equatorial* and 90° is a *polar* orbit. Any orbit between 0 and 90° is called a *posigrade* orbit and an orbit between 90 and 180° are termed a *retrograde* orbit. An important property of inclination is that the maximum ground trace latitude of a posigrade satellite is equal to the satellites inclination (inclination - 90° for the case of retrograde satellites). [Ref. 3:pp. 2.29]

1. Sun Synchronous Orbit

A special type of near circular, low earth orbit is the sun synchronous orbit. By definition, a sun synchronous satellite's orbital plane is always oriented at the same angle from the sun. A satellite in sun synchronous orbit makes use of perturbations due to the

oblateness of the Earth. The perturbative force causes the regression of the line of nodes (the orientation of the satellite plane) to match the rate that the Earth is revolving around the sun (i.e., about 360° ÷ 365.25 = 0.9856 degrees/day). Depending on altitude, a sun synchronous satellite will require an inclination of 95-105 ° to achieve the proper rate. [Ref 3:pp. 2.42-2.44]

A sun synchronous orbit can be selected so that the satellite is always passing over a given point on the Earth's surface at the same local time. This orientation is ideal for photographic missions, such as polar orbiting meteorological satellites, that require repetetive views of the same surface area under the same sun angle conditions. [Ref. 3:pp. 2.44]

2. Molniya Orbit

Another less desirable change in the orientation of orbits caused by the oblateness of the Earth is the rotation of the line of apsides. Since the line of apsides is the semi-major axis, this rotation will change the orientation of perigee and apogee. This motion, that varies in strength depending on inclination, must be compensated for by satellites that are trying to maintain their orientation over a location on the Earth's surface. The sun synchronous orbit avoided this problem by requiring a circular orbit so that perigee is an undefined quantity and for most purposes the satellite's orientation remains unchanged.

If, however, a long duration is required over the same portion of the Earth's surface either a geosynchronous orbit or an elliptical orbit with its characteristically large percentage of time spent near apogee may be used. There is a critical angle of inclination (63.43°) where the line of apsides will not change orientation [Ref. 8:pp.17]. A satellite with this inclination and a 12 hour period is commonly referred to as a *molniya* orbit, so called because it was first used by Soviet spacecraft instead of the geosynchronous orbit.

This orbit is much less expensive to achieve than geosynchronous and is also capable of providing direct overhead coverage at latitudes off the equator.

C. ORBIT CLASSIFICATION BY PERIOD

The period of a satellite is determined solely by its semi-major axis so the satellite may either be in a circular or an elliptical orbit. In general a satellite is classified according to period by dividing the orbital period into the length of the sidereal day. The result of this operation is either integer, rational, or irrational. Integer orbits exhibit ground trace patterns that repeat in 1 day cycles whereas irrational orbits do not trace any discernible pattern on the Earth's surface. Rational orbits fall somewhere between and may require many days before the orbit begins to repeat the ground trace pattern. [Ref. 5:pp. 83-99]

1. Geosynchronous Orbit (P=23 hr 56 min)

The *geosynchronous* orbit is an orbit whose period is identical to the length of the sidereal day. If the orbit has a non zero inclination the ground trace will be a figure eight that may be symmetric (circular orbits) or asymmetric (elliptic orbits) about the equator. As the inclination for a circular orbit is decreased, the satellite's orbit will trace smaller and smaller figure eights until a limiting case of the geosynchronous orbit is reached.

This orbit, the *geostationary* orbit, gives continual direct overhead coverage to one particular location on the Earth's equator from an altitude of 35,768 km. At this distance the satellite is capable of viewing about 1/3 of the Earth's surface but is incapable of providing direct line of sight coverage to the polar latitudes. [Ref. 8:pp.9]

2. Semi-synchronous Orbit (P = 11 hr 58 min)

The semi-synchronous orbit is interesting because it is a compromise between the coverage offered by the geostationary satellite versus polar coverage and much lower launch to orbit expenditures. Many new satellite systems, such as the NAVSTAR GPS

navigation satellites are using semi-synchronous orbits to take advantage of these features. Because a single satellite is no longer capable of providing continuous coverage for a specific location, NAVSTAR GPS requires 6 orbital planes of three satellites each for a total of 18 satellites. [Ref. 3:pp. 12.7-12.8]

VII. IMPLEMENTATION

A. INTRODUCTION

The purpose of building the standard library is to provide a common mechanism for processing information about orbits for many application programs. Coincident with this purpose is the desire to promote the use and further development of the library to meet the needs of its target audience. Since the intended user is the student of orbital mechanics, and is not a computer programmer or developer of a real time satellite control system, the library uses many analytic techniques which have been long since discarded by these other communities.

The designed objective of the standard library is to provide a readable collection of functions capable of supporting several orbit models ranging from elementary to "real world". To achieve a high degree of readability for code written in the C programming language is not an easy task. First, a discipline of indenting nested structures was used to more clearly demonstrate the scope of variables and expressions. Secondly, many of the "standard" C shortcuts such as nesting assignment statements inside of logical expressions or auto-incrementing variables were prohibited from the library source code. Liberal use of "white space" and header comments was used to enhance the identification of function boundaries and provide a brief synopsis of the task performed by each function. Finally, no C optimization techniques were allowed because they would require an increased level of programming sophistication on behalf of the reader.

Functions that make use of values created by other library functions use the same invocation techniques as the user. In other words, the standard library follows the philosophy of writing to a non-programmer audience by avoiding shortcuts which would

unintentionally obscure the meaning of the function from the user. The meaning of each function is therefore more important than the overhead created by this calling discipline.

The standard library was designed by making use of the programming practices of developing multiple layers of abstraction and functional decomposition. Layers of abstraction are realized by hiding the implementation of each function from the user and requiring the user to access each particular fact concerning an orbit via the standard library interface. In many cases the user is supplied with a return value unaware that the function has requested the services of other functions. Abstract data types allow the actual physical representation of the data to be removed as a user concern and enhance code readability. Decomposing the standard library into logical collections of functions make it easier for the user to trace the origin of a specific function and to narrow the scope of what must be assimilated at one time. Each section of the library is described in general terms below, while the individual routines are documented in Appendix B.

B. LIBRARY NAMING CONVENTIONS

Because C is a case sensitive programming language (i.e., "cAt" is not the same as "cat") the following capitalization conventions were adopted:

- Names appearing in ALLCAPS are constants that cannot be changed by the user.

- Names that begin with a capital letter, such as OrbitType, are C typedefs and represent the name of an abstract data type. A typedef may or may not be associated with an underlying structure that may be hidden from the user's view.

- Names that begin in lower case and end with parentheses are function calls. The standard library function names begin with two letters followed by an underscore character (e.g., xx_). The two character prefix (gl, tl, cv, kl, and cs) is an abbreviation for the name of a sub-library that is part of the standard library.

- Names that do not match any of the above descriptions are typically variable names.

Note that adherence to these naming conventions is strictly limited to code found inside of the standard library and is not enforced in user programs, such as the simulation. This is because the programming traditions for various operating systems, such as the Macintosh's, are subject to their own conventions and are often contradictory!

C. THE GENERAL LIBRARY

The library prefix "gl_" is used to define a General Library of common equations that describe the size and nature of an orbit. These functions correspond roughly to the material found in the first part of Chapter III. This library contains an extensive collection of inverse relationships because one value may be required from one of many different known parameters for an orbit. Therefore given the proper minimal set of data it is possible to obtain all of the information available for a particular orbit. To simplify equation solving in introductory orbital mechanics courses it is common for many of the orbital values to be specified without the quality of direction. The values represented by the General Library have adopted this convention and are specified as scalar rather than vector quantities.

D. THE CONVERSION LIBRARY

The Conversion Library (cv_) actually contains most of the calling discipline between the user's view of the orbital data and the internal format that the standard library uses. Since the internal representation is hidden from the user, it is important that several different methods of supplying and retreiving values be provided.

The abstract concept of angular measure, time systems, and several distance

measurement systems are all supported. For example, the user could express an orbital inclination measure in degrees and use the Conversion Library to store this value. Later the value could be retreived either in degrees or radians, without the user being aware of what format the standard library was using to compute its information. Similar conversions are accomplished between sidereal and solar time, metric and canonical distances, metric and canonical speeds, and even hours and canonical time units.

E. THE TIME LIBRARY

Time is a very important concept when considering the characteristics of a satellite. Therefore the functions that manage time values are organized into a seperate library. The commonality expressed in other libraries is much greater than what is found in the Time Library (tl_). In fact, the Time Library may be regarded as a highly specialized subset of routines that could have been located in the Conversion Library. This library provides a table lookup routine to convert known Universal and Sideral time values, a method to interpolate time values that are not included in the lookup table, a coarse time zone calculator, an algorithm for computing the local Sidereal time at an observer's location, and a pair Julian Date conversion routines.

The table lookup routine encorporates a selected value for the time difference between Universal and Sidereal time for each of the past 21 years. This value may then be used with the interpolation function to obtain an approximate time conversion for anywhere within a given year. These functions may be called directly and are also used by the local sideral time algorithm to produce its results.

The time zone calculator is useful when it is necessary to know how many hours different a local time is from Greenwich Mean Time. Successive calls to this routine could also be used to determine the number of hours that separate two local time zones. The

values returned by this function are only approximate because the actual time zones deviate from being truly longitudinal and because of differences in the local government's adoption of daylight savings time. The U.S. Uniform Time Act of 1966 established standard abbreviations for time zones that encompass U.S. territory [Ref. 9:pp. 256]. These abbreviations, as well as Greenwich Mean Time, are available to the user via another time zone calculation routine. Values in the internal table managed by this function may easily be modified to include other localized abbreviations.

A very important primitive routine for managing access to dates is the calculation of the number of days between dates or determining a date that is a given number of days away from an initial date. Although these routines could be coded directly in a "brute force" manner, the Time Library takes a different approach. By being able to convert Julian dates to and from the conventional month, day, and year values, these routines can satisfy the date calculation requirement and provide the additional service of Julian date computation. The algorithms used are numerical in approach and compensate easily for the leap year case. [Ref. 10:pp. 19-20]

F. THE COORDINATE SYSTEM LIBRARY

The libraries mentioned earlier use abstract data values that, for the most part, represent single values. The Coordinate System Library (cs_) makes a departure from this approach and instead deals with an aggregate data value, whose definition is still abstracted. The reason for this difference is to encourage the use of a predefined data type called "OrbitType". The user of this library is probably calculating satellite coordinates relative to a similar structure already and can use the predefined type without too much additional effort. Another reason to enforce this approach is that the Coordinate System Library was designed to support an abstracted notion of a satellite orbit. This principle

would have been violated if the user were required to make conversions directly from the single internal representation used by the standard library.

G. THE KEPLER PROBLEM LIBRARY

This is the smallest library (kl_) and contains all of the code that relates directly to the solution of the Kepler Problem outlined in Chapter V of this thesis. The most important calculation performed is the Newton-Raphson iteration method of root finding. This algorithm is used to determine approximate values for the Eccentric Anomaly from an initial Mean Anomaly and a specified time difference.

VIII. CONCLUSIONS AND RECOMMENDATIONS

Perhaps the best measure of the usefulness of computer software is the amount of further development it receives. The simulation implements only a large subset of the functions provided by the library and thus should not be used as an evaluative measure for routines included in the library. The simulation does attempt, however, to illustrate how many of the included routines might be used in context and provides a vehicle for guiding future library expansion.

Several areas that the library could benefit from future enhancements are speed optimization, better handling of special case orbits, increased versatility of existing routines and inclusion of new routines. The library was conceived as a method of illustrating various orbital phenomena and speed of execution was secondary to the pedagogical value of the library functions. Each routine is implemented in the same manner that the material was presented. Thus the reader may more easily follow the logic behind each function.

The library may be used in other programs that require more computational efficiency than the current implementation. Certain routines may be identified as less than optimal when used with real time graphics programs, for example. An optimized version, capable of providing faster results (perhaps with a loss in accuracy) than its counterpart, can be constructed and used instead of the existing function.

There are some orbits which are not easily calculated with the algorithms used by the standard library. Most notable among these is the ordinary geostationary orbit. The algorithm presented will not work because several Classical Orbital Elements (namely, argument of perigee and longitude of the ascending node) are undefined for this particular

orbit. Either a new algorithm could be introduced or these orbits could be handled by several special case functions. Therefore, these conditions do not represent a severe problem, but are easily handled within the framework of the existing library.

The versatility of the library could be increased in several directions, and this depends on the target audience's tastes and requirements. The primary areas of interest are systems of units and coordinate systems. For example, there may be a need for English units of measurement or a pointing vector from the position of one satellite to another. These functions may be constructed by cloning an existing function and encapsulating the user's specific requirement in a new set of functions.

The library is also an attempt at providing a basis for including future concepts concerning orbits. There are many topics that fit this category including atmospheric drag effects, calculating rendezvous problems, and determining an orbit from launch location and initial velocity. The degree of difficulty in implementing these capabilities varies widely, but should be rendered easier by the existence of the standards developed by this library.

The reasons to study the motion of satellites around the planet Earth are numerous. The mechanisms behind satellite systems, communications capabilities or simple curiosity about the way space objects behave are more clearly comprehended if they can be modeled with pictures. The standard library is a starting point for answering these questions. By using the library, the simulation is capable of demonstrating the basic orbital concepts that hopefully, will generate more sophisticated questions.

APPENDIX A SYMBOL GLOSSARY

There are many symbols and abbreviations used throughout this paper to represent specific variables, constants or concepts. This glossary is an alphabetical listing of terminology by English and Greek alphabets, subscripts and superscripts, and by coordinate system abbreviations. Many of the symbols used are considered "standard" notation, but occasionally there is a conflict in useage between different authors. Known alternative symbols used by some references are provided in square brackets ([]) following the symbol used by this thesis.

Where vector quantities are appropriate the corresponding variable name is **boldfaced** (i.e., the velocity vector **v** is a vector quantity, while the speed v is a scalar value equal to the magnitude of **v**). Where a symbol is only valid as a vector quantity (such as coordinate system abbreviations) the symbol will appear in boldface. In some cases symbols may be located under more than one listing, and the proper symbol definition is determined from the context in which the symbol is found.

ENGLISH SYMBOLS

a, a_{earth}	semimajor axis of an ellipse or semimajor axis (equatorial radius) of the Earth.
A_Z	azimuth angle measured from north clockwise to an object.
b	semiminor axis of an ellipse.
e, e_{earth}	eccentricity of an ellipse or eccentricity of a reference spheroid representing the Earth.
E	eccentric anomaly.
E_l	elevation angle measured from horizon to an object.

f	flattening of the Earth.
F, F'	foci of an ellipse where one focus (F) is occupied by the primary and the other (F') is an empty focus.
G	universal gravitational constant.
h	altitude or distance that an object is above a reference radius (r).
i	inclination angle of the orbit plane to the equatorial plane.
IJK	Geocentric Equatorial coordinate system.
$J_i^{(0)}$	ith zonal harmonic coefficient in the Earth's gravitational potential.
k	constant of gravitation for the Earth.
M_{earth}	mass of the Earth.
M, M_0	variable or constant expression for mean anomaly.
n	mean motion of a satellite.
p	semiparameter of a conic section (ellipse).
P	period of a satellite.
P_i	ith Legendre polynomial used in representing the Earth's gravitational potential.
PQW	Perifocal coordinate system.
r, r_a, r_p, r_{earth}	general radius of an ellipse, radius at apogee, radius at perigee, or radius of the Earth.
SEZ	Topocentric coordinate system.
t, t_0	time variable or arbitrary epoch time.
T	time of periapsis passage.
U	gravitational potential. [Φ]
UT	universal (solar) time.
v, v_{esc}	velocity or escape velocity of a satellite.

GREEK SYMBOLS

α	right ascension angle.
δ	declination angle.
Δ	small increment, difference.
ε	specific energy.
θ, θ_g, θ_{g0}	local sidereal time, Greenwich sidereal time or Greenwich sidereal time at epoch.
λ_E	local hour angle.
μ	gravitational parameter of a primary (GM_{earth}).
ν	true anomaly.
π	pi.
ρ	range to an object or intermediary angular value.
ρ'	angular rotation rate of the Earth. [ω_{earth}]
τ	canonical (characteristic) unit of time.
ϕ	geodetic latitude. [L]
ϕ_g	geocentric latitude. [β, ϕ']
ψ	flight path angle. [ϕ]
ω	argument of perigee.
Ω	longitude of the ascending node.

SUPERSCRIPTS

'	a derivative of some variable generally taken with respect to time.
-1	inverse of a trigonometric function.

SUBSCRIPTS

apogee	relating to the apogee position of an orbit.
earth	relating to the Earth.
esc	escape value from gravitational force.
E	east.
g	Greenwich (time) or geodetic (latitude).
n	iterative value.
perigee	relating to the perigee position of an orbit.
0	epoch or initial value.
ω	referring to the orbit plane.

COORDINATE SYSTEM ABBREVIATIONS

IJK	geocentric equatorial coordinates : x,y,z.
PQW	perifocal coordinates : $x_\omega, y_\omega, z_\omega$.
SEZ	topocentric coordinates : A_z, E_l, ρ.
$\phi\lambda$h	geographic coordinates: ϕ, λ, h.
R.A.-Decl.	right ascension declination coordinates: α, δ, r.

APPENDIX B

STANDARD LIBRARY SOURCE LISTING

```
/*********************************************************************
          FILENAME              :  StdLib.h
          DESCRIPTION           :  header file for StdLib.c
          ENVIRONMENT           :  Macintosh SE 1Mb RAM
                                   LightSpeed™ C v2.15
          AUTHOR                :  Captain Kenneth L. BEUTEL USMC
          ADVISORS              :  Prof. Dan Davis
                                   Prof. Dan Boger
                                   Naval Postgraduate School, Monterey CA
          REMARKS               :  none
          VERSION               :  0.9 (3/6/88)

          CHANGES               :  3/6/88 Formatted for MacWrite conversion

********************************************************************* */
/* typedef and structure declarations                                  */
typedef   double    Angle;    /* radian  angular measure               */
typedef   double    Time;     /* decimal seconds time of day measure   */
typedef   double    Dist;     /* distance in kilometers                */
typedef   double    Real;     /* numeric format for other non-ints     */

typedef struct
   {
      Real          date;     /* julian day and year                   */
      Time          time;     /* time in decimal hours                 */
   } DateTime;

typedef struct
   {
      char          name[20]; /* identity of this orbit                */
      Dist          semimajor;
                              /* semimajor axis (a)                    */
      Real          eccentricity;
      Angle         inclination;
      Angle         mean_anom;/* Mean anomaly (M0)                     */
      Angle         arg_of_perigee;
      Angle         long_of_asc_node;
                              /* longitude of ascending node CapOMEGA */
      Time          epoch;    /* epoch time (t)                        */
      Real          date;     /* epoch julian date                     */
   /* ******** The following are for internal use by library routines
      ******** only.  The user should access these values via function
      ******** calls.                                                  */
```

```
      Angle          eccentric_anom;
                                /* eccentric anomaly at time T          */
   } OrbitType;

typedef struct
   {
   /* Perifocal (PQW) position coordinates                               */
      Dist           x;         /* positive x points towards perigee    */
      Dist           y;         /* pos y in orbit plane 90° ahead of x  */
      Dist           z;         /*  z is positive normal to orbit z= 0  */
   } PQWCoord;

typedef struct
   {
   /* Geocentric Equatorial (IJK) position coordinates :                 */
      Dist           x;         /* positive x points to vernal equinox  */
      Dist           y;         /* positive y 90° R.H of x              */
      Dist           z;         /* positive z normal to rotation (north)*/
   } IJKCoord;

typedef struct
   {
   /* Topocentric Horizon (SEZ) position coordinates :                   */
      Angle          elevation;/* clockwise from -s                     */
      Angle          azimuth;  /* positive upwards from horizontal      */
      Dist           range;    /* from observer to satellite            */
   } SEZCoord;

typedef struct
   {
   /* Right Ascension-Declination (R.A.-D.) position coordinates :       */
      Dist           r;         /* distance from primary's center       */
      Angle          ra;        /* rt. ascension angle from ver equinox */
      Angle          decl;      /* declination from celestial equator   */
   } RADCoord;

typedef struct
   {
   /* Geographic (GEO) position coordinates :                            */
      Dist           altitude; /* distance above reference ellipsoid    */
      Angle          latitude; /* angle above or below the equator      */
      Angle          longitude;/* angle east of Greenwich meridian      */
   } GEOCoord;

typedef struct
   {
      char           name[20]; /* identity of primary (normally earth) */
      Real           curr_time;/* time in decimal days since ephemeris */
      Real           grav_param;
                                /* gravitational parameter (GM)         */
```

```
    Dist          radius;   /* mean equatorial radius of primary    */
    Time          herg;     /* base time unit                       */
    Real          speed;    /* base time rate of change dist        */
    Real          ang_rot;  /* ang rotation rate of earth           */
    Real          eccentricity;
                            /* oblateness of reference spheroid     */
    Real          J2;       /* 2nd zonal hgarmonic                  */
    Real          J4;       /* 4th zonal harmonic                   */
} PrimaryType;


/* *************** extern variables found in stdlib.c *************** */
extern PrimaryType  gl_primary;


/* gl_ *************** prototypes for functions in general library    */
void      gl_idflt();         /* init default values for primary      */
char      *gl_gpnam();        /* get primary's name                   */
void      gl_iorbt();         /* set initial values for orbit         */
Dist      gl_gorba();         /* get orbital semimajor axis           */
char      *gl_gorbn();        /* get orbital name                     */
Real      gl_gorbe();         /* get orbital eccentricity             */
Angle     gl_gorbi();         /* get orbital inclination              */
Angle     gl_gorbm();         /* get orbital mean anomaly             */
Angle     gl_gorbp();         /* get orbital arg of perigee           */
Angle     gl_gorbl();         /* get orbital long of asn node         */
Time      gl_gorbt();         /* get orbital epoch          */
Dist      gl_gradp();         /* get radius of perigee                */
Dist      gl_grada();         /* get radius of apogee                 */
Dist      gl_gradi();         /* get radius at true anomaly angle     */
Dist      gl_grade();         /* get radius at Eccen Anom angle       */
Angle     gl_gtrue();         /* get true anomaly at radius           */
Dist      gl_gxpos();         /* get x pos in PQW coords at true anom */
Dist      gl_gypos();         /* get y pos in PQW coords at true anom */
Dist      gl_gsemi();         /* get semi parameter of a conic        */
Real      gl_gmean();         /* get mean motion constant             */
Time      gl_gperd();         /* get period                           */
Real      gl_gvelo();         /* get velocity at radius r             */
Real      gl_gvesc();         /* get escape velocity at radius r      */
Angle     gl_gfltp();         /* get flight path angle (0 to pi/2)    */
Dist      gl_glosd();         /* get line of sight dist to horizon    */
Dist      gl_ggswi();         /* get geometric swath width            */
Real      gl_gangm();         /* get angular momentum                 */
Real      gl_gspen();         /* get specific energy                  */

/* tl_ *************** prototypes for functions in time library       */
Real      tl_gzone();         /* get time zone number past GMT        */
char      *tl_gznam();        /* get time zone name from time zone no */
Time      tl_gknow();         /* table lookup known Greenwich time    */
Angle     tl_ggren();         /* get Greenwich sidereal time (approx) */
```

```
Angle     tl_glstm();          /* get local sidereal time at longitude */
Real      tl_gjuld();          /* get julian date (in whole days)      */
void      tl_gmdyr();          /* get month,day and year from jul date */

/* cv_ *************** prototypes for functions in conversion library */
Angle     cv_sangd();          /* set angle in degrees                 */
Angle     cv_sangr();          /* set angle in radians                 */
Real      cv_gangd();          /* get angle in degrees                 */
Real      cv_gangr();          /* get angle in radians                 */
Time      cv_gsolt();          /* get solar time from sidereal time    */
Time      cv_gsidt();          /* get sidereal time from solar time    */
Dist      cv_cdisk();          /* convert canonical distance to km     */
Dist      cv_ckdis();          /* convert km to canonical distance     */
Real      cv_ckspd();          /* convert Km/sec to canonical Speed    */
Real      cv_cspdk();          /* convert canonical Speed to Km/sec    */
Time      cv_chtim();          /* Convert secs to canonical TIMe       */
Time      cv_ctimh();          /* Convert canonical TIMe to secs       */

/* cs_ *************** prototypes for functions in Coord Sys Trans lib*/
void      cs_gpqwc();          /* get PQW coordinates       (trans #1) */
void      cs_gijkc();          /* get IJK coordinates       (trans #2) */
void      cs_gpqwk();          /* get PQW from Keplerian    (trans #3) */
void      cs_gijks();          /* get IJK from SEZ          (trans #4) */
void      cs_gijkr();          /* get IJK from RA-Decl      (trans #5) */
void      cs_gradc();          /* get RA-Decl coordinates   (trans #6) */
void      cs_ggeoc();          /* get GEO coordinates       (trans #7) */
void      cs_gijkg();          /* get IJK from GEO          (trans #8) */

/* kl_ *************** prototypes for functions in Kepler Library     */
Angle     kl_gecca();          /* get eccentric anomaly at time t      */
Angle     kl_geccp();          /* perturbed eccentric anom at time t   */


/* ******************* macros expand calls to variables that are located
   in the general library using shorthand that is commonly found
   in orbit equations.                                                 */
#ifndef   PI
          #define    PI                  3.14159265359
#endif

#ifndef   MU
          #define    MU                  gl_primary.grav_param
#endif

/* numerical convergence value                                        */
#ifndef   GL_EPSILON
          #define    GL_EPSILON          0.000001
#endif
```

```
/*********************************************************************

          FILENAME              :  genLib.c
          DESCRIPTION           :  General Purpose StdLib source file (gl_)
          ENVIRONMENT           :  Macintosh SE 1Mb RAM
                                   LightSpeed™ C v2.15
          AUTHOR                :  Captain Kenneth L. BEUTEL USMC
          ADVISORS              :  Prof. Dan Davis
                                   Prof. Dan Boger
                                   Naval Postgraduate School, Monterey CA
          REMARKS               :  none
          VERSION               :  0.9 (3/6/88)

          CHANGES               :  3/6/88 Formatted for MacWrite conversion

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

#include <stdio.h>
#include <storage.h>
#include <math.h>
#include <strings.h>

/* include the StdLib header file's typedefs and global variables     */
#include "StdLib.h"


/* ************* global variables that must be initialized for genlib */
PrimaryType          gl_primary;
                              /* declaration of internal primary name */

/* Because this is a library no main procedure is allowed or required.
   all function names follow the standard convention :
          gl_adddd - where gl   is the library name (General Library)
          a          is the action that the function performs (such as
                     initialize, get, set, and query)
          dddd is the function's descriptive name                     */


/* ********************************************************************
   gl_idflt : initialize default values (Bate et.al.) for  earth as a
              primary in metric units
   ******************************************************************** */
   void
gl_idflt()
{  /* Note that values for primary must be specified in metric units  */
   strcpy(gl_primary.name, "Earth");
   gl_primary.grav_param = 3.986012e5;
                              /* in kg                                */
   gl_primary.radius = 6378.145;
                              /* in km                                */
```

```
   gl_primary.herg = 806.8118744;
                              /* in sec                                  */
   gl_primary.speed = 7.90536828;
                              /* in km/sec                               */
   gl_primary.ang_rot = 7.29211586e-5 * 3600.0;
                              /* in rad/hr                               */
   gl_primary.eccentricity = 0.08182;
   gl_primary.J2 = 1082.64e-6;
   gl_primary.J4 = -2.5e-6;
}   /* ********** function initialize defaults ********************** */


/* ********************************************************************
   gl_gpnam : gets the name of the primary
   ******************************************************************** */
   char *
gl_gpnam()
{
   return(gl_primary.name);
}   /* ********** function get primary name ************************** */



/* ******* functions that work with the Orbit Typedef records ******* */

/* ********************************************************************
   gl_sorbt : set values for new orbit record
   ******************************************************************** */
   void
gl_sorbt(name, a, e, i, mean_anom, peri, asn, epoch, date, orbrec)
   char              *name;
   Real              a, e;
   Angle             i, mean_anom, peri, asn;
   Time              epoch;
   Real              date;
   OrbitType         *orbrec;
{
   strcpy(orbrec->name, name);
   orbrec->semimajor = a;
   orbrec->eccentricity = e;
   orbrec->inclination = i;
   orbrec->mean_anom = mean_anom;
   orbrec->arg_of_perigee = peri;
   orbrec->long_of_asc_node = asn;
   orbrec->epoch = epoch;
   orbrec->date = date;

   orbrec->eccentric_anom = 0.0;
   kl_gecca(&*orbrec, epoch); /* initial calc of ecc anomaly          */
```

```
}   /* ****************** function set orbit ************************ */


/* ***********************************************************************
   gl_gorbn :  get orbit name from orbit record
   ********************************************************************** */
   char   *
gl_gorbn(orbit)
   OrbitType         orbit;
{
   return(orbit.name);
}   /* ********** function get orbit name *************************** */


/* ***********************************************************************
   gl_gorba :  get orbit semimajor axis from orbit record
   ********************************************************************** */
   Dist
gl_gorba(orbit)
   OrbitType         orbit;
{
   return( orbit.semimajor );
}   /* ********** function get orbit semimajor axis ***************** */


/* ***********************************************************************
   gl_gorbe :  get orbit eccentricity from orbit record
   ********************************************************************** */
   Real
gl_gorbe(orbit)
   OrbitType         orbit;
{
   return( orbit.eccentricity );
}   /* ********** function get orbit eccentricity ******************* */


/* ***********************************************************************
   gl_gorbi :  get orbit inclination from orbit record
   ********************************************************************** */
   Angle
gl_gorbi(orbit)
   OrbitType         orbit;
{
   return( orbit.inclination );
}   /* ********** function get orbit inclination ******************** */


/* ***********************************************************************
   gl_gorbp :  get orbit argument of perigee from orbit record
   ********************************************************************** */
```

```
   Angle
gl_gorbp(orbit)
   OrbitType         orbit;
{
   return( orbit.arg_of_perigee );
}   /* ********** function get argument of perigee ****************** */


/* ********************************************************************
   gl_gorbm :  get orbit mean anomaly from orbit record
   ******************************************************************** */
   Angle
gl_gorbm(orbit)
   OrbitType         orbit;
{
   return( orbit.mean_anom );
}   /* ********** function get orbit mean anomaly ******************* */


/* ********************************************************************
   gl_gorbl :  get orbit longitude of ascending node from orbit record
   ******************************************************************** */
   Angle
gl_gorbl(orbit)
   OrbitType         orbit;
{
   return( orbit.long_of_asc_node );
}   /* ********** function get long of asc node  ******************** */


/* ********************************************************************
   gl_gorbt :  get orbit epoch time from orbit record
   ******************************************************************** */
   Time
gl_gorbt(orbit)
   OrbitType         orbit;
{
         return(orbit.epoch);
}   /* ********** function get epoch time *************************** */


/* ********************************************************************
   gl_gradp :  get radius of perigee
   ******************************************************************** */
   Dist
gl_gradp(a,e)
   Real              a,e;
{
   return( a * (1.0 - e) );
}   /* ********** function get radius of perigee ******************** */
```

```
/* ************************************************************************
   gl_gorba :  get radius of apogee
   ************************************************************************ */
   Dist
gl_grada(a,e)
   Real              a,e;
{
   return(a * (1.0 + e));
}   /* ********** function get radius of apogee ******************** */


/* ************************************************************************
   gl_gradi :  get radius at true anomaly angle nu
   ************************************************************************ */
   Dist
gl_gradi(p,e,nu)
   Real              p,e;
   Angle             nu;
{                                /* p = a (1-e*e) semiparameter          */
   return ( p / (1.0 + e * cos( cv_gangr(nu) )) );
}   /* ***** function get radius at true anomaly angle ************** */


/* ************************************************************************
   gl_grade :  get radius at Eccentric anomaly angle E
   ************************************************************************ */
   Dist
gl_grade(a,e,eccen_anom)
   Dist              a;
   Real              e;
   Angle             eccen_anom;
{                                /* from Smith Pp. 65          */
   return (a * (1.0 - e * cos( cv_gangr(eccen_anom) )));
}   /* ***** function get radius at Eccentric anomaly angle ********* */


/* ************************************************************************
   gl_gtrue :  get true anomaly angle nu at radius r
   ************************************************************************ */
   Angle
gl_gtrue(x,y,p,e)
   Dist              x,y,p;
   Real              e;
{                                /* p = a (1-e*e) semiparameter          */
Dist                 r;
Angle                nu;
```

```
   if (e == 0.0)              /* divide by zero error check          */
      return( 0.0);

   r = sqrt( x*x + y*y);
   nu = acos( ((p/r) - 1.0) / e );
   if ((x>0) && (y>0))
      ;                       /* nu = nu                             */
   else if ((x<0) && (y>0))
      ;                       /* nu = (PI/2.0) + nu                  */
   else if ((x<0) && (y<0))
      nu = (2.0*PI) - nu;
   else if ((x>0) && (y<0))
      nu = (2.0*PI) - nu;

   return ( nu );
}   /* ***** function get true anomaly angle ********************** */


/* ********************************************************************
   gl_gxpos :  get x position in PQW coords at true anom
   ******************************************************************** */
   Dist
gl_gxpos(p,e,nu)
   Dist             p;
   Real             e;
   Angle            nu;
{                               /* p = a (1-e*e) semiparameter         */
   return( (p * cos( cv_gangr(nu) )) / (1.0 + e * cos( cv_gangr(nu) )) );
}   /* **** function get x position in PQW coords at true anom ****** */


/* ********************************************************************
   gl_gypos :  get y position in PQW coords at true anom
   ******************************************************************** */
   Dist
gl_gypos(p,e,nu)
   Dist             p;
   Real             e;
   Angle            nu;
{                               /* p = a (1-e*e) semiparameter         */
   return( (p * sin( cv_gangr(nu) )) / (1.0 + e * cos( cv_gangr(nu) )) );
}   /* **** function get y position in PQW coords at true anom ****** */


/* ********************************************************************
   gl_gvelo :  get velocity at radius r
   ******************************************************************** */
   Real
gl_gvelo(r,a)
   Dist             r,a;
```

```
{                               /* note MU = 1.0 for canonical units    */
        return( sqrt(MU * ((2.0/r) - (1.0/a))) );
}   /* **** function get velocity at r ****************************** */


/* ************************************************************************
   gl_gvesc :  get escape velocity at radius r
   ************************************************************************ */
   Real
gl_gvesc(r)
   Real             r;
{                               /* note MU = 1.0 for canonical units    */
   return( sqrt(MU * (2.0/r)) );
}   /* **** function get escape velocity **************************** */


/* ************************************************************************
   gl_gfltp :  get flight path angle at radius r, using h,v
   ************************************************************************ */
   Angle
gl_gfltp(h, r, v)
   Real             h;
   Dist             r;
   Real             v;
{
   return( (Angle) acos(h / (r * v)) );
}   /* **** function get flight path angle at r ********************** */


/* ************************************************************************
   gl_glosd :  get line of sight dist to horizon at radius r,
               using gamma (angle above horizon)
   ************************************************************************ */
   Dist
gl_glosd(r, gamma)
   Dist             r;
   Angle            gamma;
{
Real                temp;

   temp = gl_primary.radius * cos( cv_gangr(gamma) ) / r;
   return( r * cos(cv_gangr(gamma) + asin(temp))/cos(cv_gangr(gamma)) );
}   /* **** function get line of sight dist to horizon ************** */


/* ************************************************************************
   gl_ggswi :  get geometric swath width
   ************************************************************************ */
   Dist
gl_ggswi(r)
```

```
   Dist              r;
{
   return( 2.0 * gl_primary.radius * acos (gl_primary.radius / r) );
}   /* ***** function get geometric swath width ********************* */


/* ***************** O R B I T A L   C O N S T A N T S ****************
         Constants need only be evaluated once per orbit
   ************************ ORBITAL CONSTANTS ************************ */

/* ********************************************************************
   gl_gsemi :  get semi parameter
   ******************************************************************** */
   Dist
gl_gsemi(a,e)
   Dist              a;
   Real              e;
{
   return( a *(1.0 - e*e) );
}   /* ******* function get semiparameter of conic ****************** */


/* ********************************************************************
   gl_gangm :  get angular momentum
   ******************************************************************** */
   Real
gl_gangm(a, e)
   Dist              a;
   Real              e;
{                              /* evaluate r x v at perigee            */
Real                 rperi, vperi, p;

   p = gl_gsemi(a, e);
   rperi = gl_gradi(p, e, 0.0);
   vperi = gl_gvelo(rperi, a);
   return( rperi * vperi );    /* flt path angle is 90 degrees         */
}   /* ***** function get angular momentum ************************** */


/* ********************************************************************
   gl_gmean :  get mean motion constant
   ******************************************************************** */
   Real
gl_gmean(a)
   Dist              a;
{                              /* MU in canonical units = 1.0 by defn  */
   return( sqrt(MU) / pow(a,1.5));
                               /*     m = ( MU/a*a*a) ^1/2             */
}   /* ******* function get mean motion constant ******************** */
```

```
/* ************************************************************************
   gl_gperd :  get period
   ************************************************************************ */
   Time
gl_gperd(n)
   Real              n;
{
   return( 2 * PI / n );
}   /* ******* function get period ********************************** */


/* ************************************************************************
   gl_gspen :  get specific energy
   ************************************************************************ */
   Real
gl_gspen(v,r)
   Real              v;
   Dist              r;
{
   return( (v * v / 2.0) + (MU / r) );
}   /* ****** function get specific energy ************************** */
```

```
/*********************************************************************

          FILENAME             :  convert.c (cv_)
          DESCRIPTION          :  Conversion formula StdLib source file
          ENVIRONMENT          :  Macintosh SE 1Mb
                                  LightSpeed™ C v2.15
          AUTHOR               :  Captain Kenneth L. BEUTEL USMC
          ADVISORS             :  Prof. Dan Davis
                                  Prof. Dan Boger
                                  Naval Postgraduate School, Monterey CA
          REMARKS              :  none
          VERSION              :  0.9 (3/6/88)

          CHANGES              :  3/6/88 Formatted for MacWrite conversion

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

#include <stdio.h>
#include <storage.h>
#include <math.h>
#include <strings.h>

/* include the StdLib header file's typedefs and global variables     */
#include "StdLib.h"



/* Because this is a library no main procedure is allowed or required.
   all function names follow the standard convention :
          cv_adddd - where
                     cv is the library name, (cONvERSION LIBRARY)
                      a  is the action that the function performs (such as
                           (i)initialize, (g)get, (s)set, and (c)convert)
                           dddd is the function's descriptive name     */
/* ******************************************************************
   SPECIAL NOTE : all angular measures are stored internally in radians
                  all distance quantities are stored in kilometers
                  all time and mass quantities are stored in the initial
                       units specified
   ****************************************************************** */



/* ******************************************************************
   cv_sangd : set angle measure in degrees
   ****************************************************************** */
   Angle
cv_sangd(degrees)
   Real              degrees;
{
```

```
   return( 0.0174532925199 * degrees );
}    /* *************** function set angle (degrees) *************** */


/* ************************************************************************
   cv_sangr : set angle measure in radians
   ************************************************************************ */
   Angle
cv_sangr(radians)
   Real              radians;
{
   return( radians );
}   /* *************** function set angle (radians) ***************** */


/* ************************************************************************
   cv_gangd : get angle measure in degrees
   ************************************************************************ */
   Real
cv_gangd(value)
   Angle             value;
{
   return( 57.2957795131 * value );
}   /* *************** function set angle (degrees) ***************** */


/* ************************************************************************
   cv_gangr : get angle measure in radians
   ************************************************************************ */
   Real
cv_gangr(value)
   Angle             value;
{
   return( (Real) value );
}  /* *************** function set angle (radians) ***************   */


/* ************************************************************************
   cv_gsolt : get solar time from sidereal time
   ************************************************************************ */
   Time
cv_gsolt(value)
   Time              value;
{
   return( value * 1.0027379093 );
                              /* = sideral * sol/sid day              */
}   /* *************** function get solar time ********************** */


/* ************************************************************************
```

```
   cv_gsidt : get sidereal time from solar time
   ********************************************************************* */
   Time
cv_gsidt(value)
   Time              value;
{
   return( value * 0.9972695664 );
                              /* = solar * sid/sol day                  */
}   /* *************** function get solar time ********************** */


/* ********************************************************************
   cv_ckdis : convert Kilometers to canonical DIStance
   ********************************************************************* */
   Dist
cv_ckdis(km_distance)
   Dist              km_distance;
{
   return( km_distance / gl_primary.radius );
}   /* *********** function convert canonical dist from km ********** */


/* ********************************************************************
   cv_cdisk : convert canonical DIStance to Kilometers
   ********************************************************************* */
   Dist
cv_cdisk(canon_distance)
   Dist              canon_distance;
{
   return( canon_distance * gl_primary.radius );
}   /* *********** function convert km from canonical dist ********** */


/* ********************************************************************
   cv_ckspd : Convert Km/sec to canonical Speed (magnitude of velocity)
   ********************************************************************* */
   Real
cv_ckspd(km_sec)
   Real              km_sec;
{
   return( km_sec / gl_primary.speed );
}   /* ********** function get canonical speed from km/sec ********** */


/* ********************************************************************
   cv_cspdk : Convert canonical Speed (magnitude of velocity) to KM/sec
   ********************************************************************* */
   Real
cv_cspdk(canon_speed)
   Real              canon_speed;
```

```
{
   return( canon_speed * gl_primary.speed );
}   /* ************* function get km from canonical dist ************ */


/* ********************************************************************
   cv_chtim : Convert seconds to canonical TIMe
   ******************************************************************** */
   Time
cv_chtim(sec_time)
   Time              sec_time;
{
   return( sec_time / gl_primary.herg );
}   /* *********** function Convert secs to canonical Time ********** */


/* ********************************************************************
   cv_ctimh : Convert canonical TIMe to seconds
   ******************************************************************** */
   Time
cv_ctimh(canon_time)
   Time              canon_time;
{
   return( canon_time * gl_primary.herg );
}   /* ********** function Convert canonical Time to seconds ******** */
```

```
/*********************************************************************

          FILENAME              :  coordsys.c (cs_)
          DESCRIPTION           :  Coordinate Transformations source file
          ENVIRONMENT           :  Macintosh SE 1Mb RAM
                                   LightSpeed™ C v2.15
          AUTHOR                :  Captain Kenneth L. BEUTEL USMC
          ADVISORS              :  Prof. Dan Davis
                                   Prof. Dan Boger
                                   Naval Postgraduate School, Monterey CA
          REMARKS               :  none
          VERSION               :  0.9 (3/6/88)

          CHANGES               :  3/6/88 Formatted for MacWrite conversion

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

#include <stdio.h>
#include <storage.h>
#include <math.h>
#include <strings.h>

/* include the StdLib header file's typedefs and global variables     */
#include "StdLib.h"


/* Because this is a library no main procedure is allowed or required.
   all function names follow the standard convention :
          cs_adddd - where
              ct is the library name (Coordinate Transformation)
              a    is the action that the function performs (such as
                     (i)initialize, (g)get, (s)set, and (c)convert)
              dddd is the function's descriptive name                  */


/* ********************************************************************
   cs_gpqwc : get PQW coordinates from OrbitRec (transformation #1)
   ******************************************************************** */
   void
cs_gpqwc(theorbit, pqw_pos)
   OrbitType        theorbit;
   PQWCoord         *pqw_pos;
{
   pqw_pos->x = theorbit.semimajor *
              ( cos(theorbit.eccentric_anom) - theorbit.eccentricity );
   pqw_pos->y = theorbit.semimajor * sin(theorbit.eccentric_anom) *
              sqrt( 1.0 - theorbit.eccentricity * theorbit.eccentricity );
   pqw_pos->z = 0.0;/* by definition                              */
}   /* ****************** function get PQW Coords ***************** */
```

```
/* ********************************************************************
   cs_gijkc : get IJK coordinates from OrbitRec (transformation #2)
   ******************************************************************** */
   void
cs_gijkc(theorbit, ijk_pos)
   OrbitType         theorbit;
   IJKCoord          *ijk_pos;
{
PQWCoord             pqw;      /* need to get pqw first                */
Real                 cos_p, sin_p, cos_a, sin_a, cos_i, sin_i;
                               /* storage to save transcendental calcs */

   cs_gpqwc(theorbit, &pqw);
   cos_p = cos(theorbit.arg_of_perigee);
   sin_p = sin(theorbit.arg_of_perigee);
   cos_a = cos(theorbit.long_of_asc_node);
   sin_a = sin(theorbit.long_of_asc_node);
   cos_i = cos(theorbit.inclination);
   sin_i = sin(theorbit.inclination);

   ijk_pos->x = pqw.x * ( (cos_p*cos_a) - (sin_p*sin_a*cos_i) )
              + pqw.y * ( (-sin_p*cos_a) - (cos_p*sin_a*cos_i) );

   ijk_pos->y = pqw.x * ( (cos_p*sin_a) + (sin_p*cos_a*cos_i) )
              + pqw.y * ( (-sin_p*sin_a) + (cos_p*cos_a*cos_i) );

   ijk_pos->z = pqw.x * ( sin_p*sin_i )
              + pqw.y * ( cos_p*sin_i );
}   /* ****************** function get IJK Coords ***************** */


/* ********************************************************************
   cs_gpqwk : get PQW coordinates from IJK Coords and Keplerian angles
   ******************************************************************** */
   void
cs_gpqwk(theorbit, ijk_pos, pqw_pos)
   OrbitType         theorbit;
   IJKCoord          ijk_pos;
   PQWCoord          *pqw_pos;
{
Real                 cos_p, sin_p, cos_a, sin_a, cos_i, sin_i;
                               /* storage to save transcendental calcs */

   cos_p = cos(theorbit.arg_of_perigee);
   sin_p = sin(theorbit.arg_of_perigee);
   cos_a = cos(theorbit.long_of_asc_node);
   sin_a = sin(theorbit.long_of_asc_node);
   cos_i = cos(theorbit.inclination);
   sin_i = sin(theorbit.inclination);
```

```
   pqw_pos->x  = ijk_pos.x * ( (cos_p*cos_a) - (sin_p*sin_a*cos_i) )
               + ijk_pos.y * ( (cos_p*sin_a) + (sin_p*cos_a*cos_i) )
               + ijk_pos.z * ( sin_p*sin_i );

   pqw_pos->y  = ijk_pos.x * ( (-sin_p*cos_a) - (cos_p*sin_a*cos_i) )
               + ijk_pos.y * ( (-sin_p*sin_a) + (cos_p*cos_a*cos_i) )
               + ijk_pos.z * ( cos_p*sin_i );

   pqw_pos->z  = ijk_pos.x * ( sin_a*sin_i )
               + ijk_pos.y * ( -cos_a*sin_i )
               + ijk_pos.z * ( cos_i );
                             /* z should equal zero for pqw Coords   */
}   /* ************* function get PQW from Kepler/IJK *************** */


/* ****************************************************************
   cs_gijks : get IJK coordinates from SEZ Coords
   **************************************************************** */
   void
cs_gijks(theorbit, sez_pos, latitude, longitude, ijk_pos)
   OrbitType         theorbit;
   SEZCoord          sez_pos;
   Angle             latitude; /* latitude of the observer            */
   Angle             longitude;/* longitude of the observer           */
   IJKCoord          *ijk_pos;
{
DateTime             datetime; /* date time of the observation        */
Real                 cos_El, sin_El, cos_Az, sin_Az;
                             /* storage to save transcendental calcs */
Real                 cos_Lat, sin_Lat, cos_Lon, sin_Lon;
                             /* storage to save transcendental calcs */
Real                 rhoS, rhoE, rhoZ;
                             /* range expressed in SEZ coordinates   */
Angle                lst;    /* local sideral time                   */


   cos_El = cos( cv_gangr(sez_pos.elevation));
   sin_El = sin( cv_gangr(sez_pos.elevation));
   cos_Az = cos( cv_gangr(sez_pos.azimuth));
   sin_Az = sin( cv_gangr(sez_pos.azimuth));

   rhoS = - sez_pos.range * cos_El * cos_Az;
   rhoE =   sez_pos.range * cos_El * sin_Az;
   rhoZ =   sez_pos.range * sin_El;

   datetime.date = theorbit.date;
   datetime.time = theorbit.epoch;
   lst = tl_glstm(longitude, datetime);
```

```
   cos_Lat = cos( cv_gangd(latitude));
   sin_Lat = sin( cv_gangd(latitude));
   cos_Lon = cos( cv_gangd(lst));
   sin_Lon = sin( cv_gangd(lst));

   ijk_pos->x  = rhoS * (sin_Lat*cos_Lon)
               + rhoE * (-sin_Lon)
               + rhoZ * (cos_Lat*cos_Lon);

   ijk_pos->y  = rhoS * (sin_Lat*sin_Lon)
               + rhoE * (cos_Lon)
               + rhoZ * (cos_Lat*sin_Lon);

   ijk_pos->z  = rhoS * (-cos_Lat)
               + rhoZ * (sin_Lat);

}   /* ***************** function get IJK from SEZ ****************** */


/* ********************************************************************
   cs_gijkr : get IJK coordinates from Right Ascension-Declination Coords
   ******************************************************************** */
   void
cs_gijkr(theorbit, rad_pos, ijk_pos)
   OrbitType         theorbit;
   RADCoord          rad_pos;
   IJKCoord          *ijk_pos;
{
   ijk_pos->x = rad_pos.r * cos(rad_pos.decl) * cos(rad_pos.ra);

   ijk_pos->y = rad_pos.r * cos(rad_pos.decl) * sin(rad_pos.ra);

   ijk_pos->z = rad_pos.r * sin(rad_pos.decl);
}   /* ******* function get IJK Coords from Right Asc-Declin ******** */


/* ********************************************************************
   cs_gradc : get right ascension declination coordinates from IJK
   ******************************************************************** */
   void
cs_gradc(theorbit, ijk_pos, rad_pos)
   OrbitType         theorbit;
   IJKCoord          ijk_pos;
   RADCoord          *rad_pos;
{
Angle                temp_decl;

   rad_pos->r    = sqrt(ijk_pos.x*ijk_pos.x + ijk_pos.y*ijk_pos.y
                        + ijk_pos.z*ijk_pos.z);
   rad_pos->ra   = atan2(ijk_pos.y, ijk_pos.x);
```

```
   temp_decl = atan2(ijk_pos.z,
                     sqrt(ijk_pos.x*ijk_pos.x + ijk_pos.y*ijk_pos.y));
   if ((temp_decl > PI/2.0) && (temp_decl <= PI))
      temp_decl = PI - temp_decl;
   else if ((temp_decl > PI) && (temp_decl <= 1.5*PI))
      temp_decl = PI - temp_decl;
   else if (temp_decl > 1.5*PI)
      temp_decl = temp_decl - 2.0*PI;


   rad_pos->decl = temp_decl;
}   /* ***************** function get RightAsc-Decl ****************** */


/* ************************************************************************
   cs_ggeoc : get GEO coordinates from Orbit data
   ************************************************************************ */
   void
cs_ggeoc(theorbit, since_epoch, geo_pos)
   OrbitType         theorbit;
   Time              since_epoch;
                              /* time past epoch in hours             */
   GEOCoord          *geo_pos;
{
Real                 x_earth, y_earth, z_earth;
                              /* storage for intermediate values      */
Real                 rho, cos_rho, sin_rho, r;
                              /* storage to save transcendental calcs */
IJKCoord             ijk;     /* need this value to begin             */
DateTime             datetime;

   cs_gijkc(theorbit, &ijk);
   datetime.date = theorbit.date;
   datetime.time = since_epoch;
   rho = tl_ggren(datetime);  /* angle the earth has rotated thru     */
   cos_rho = cos(rho);
   sin_rho = sin(rho);

   x_earth = ijk.x *  cos_rho + ijk.y * sin_rho;
   y_earth = ijk.x * -sin_rho + ijk.y * cos_rho;
   z_earth = ijk.z;

   r = sqrt(x_earth*x_earth + y_earth*y_earth + z_earth*z_earth);

   geo_pos->latitude  = asin(z_earth / r);
   geo_pos->longitude = atan2(y_earth,x_earth) + PI;
                              /* PI/2 is correction to get east long. */
   geo_pos->altitude  = r - gl_primary.radius;

}   /* ****************** function get GEO Coords ****************** */
```

```
/* ********************************************************************
   cs_gijkg : get IJK coordinates from GEO (geocentric) and Orbit
   ******************************************************************** */
   void
cs_gijkg(theorbit, geo_pos, ijk_pos)
   OrbitType        theorbit;
   GEOCoord         geo_pos;
   IJKCoord         *ijk_pos;
{

   ijk_pos->x = (geo_pos.altitude + gl_primary.radius)
                * cos(geo_pos.latitude) * cos(geo_pos.longitude);

   ijk_pos->y = (geo_pos.altitude + gl_primary.radius)
                * cos(geo_pos.latitude) * sin(geo_pos.longitude);

   ijk_pos->z = (geo_pos.altitude + gl_primary.radius)
                * sin(geo_pos.latitude);
}   /* *************** function get IJK from GEO *****************  */
```

```
/*********************************************************************

          FILENAME              :  kepler.c
          DESCRIPTION           :  iterative Kepler Problem sol'n file (gl_)
          ENVIRONMENT           :  Macintosh SE 1Mb RAM
                                   LightSpeed™ C v2.15
          AUTHOR                :  Captain Kenneth L. BEUTEL USMC
          ADVISORS              :  Prof. Dan Davis
                                   Prof. Dan Boger
                                   Naval Postgraduate School, Monterey CA
          REMARKS               :  none
          VERSION               :  0.9 (3/6/88)

          CHANGES               :  3/6/88 Formatted for MacWrite conversion

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

#include <stdio.h>
#include <storage.h>
#include <math.h>
#include <strings.h>

/* include the StdLib header file's typedefs and global variables     */
#include "StdLib.h"

/* Because this is a library no main procedure is allowed or required.
   all function names follow the standard convention :
          gl_adddd - where
             gl   is the library name (General Library)
             a    is the action that the function performs (such as
                    initialize, get, set, and query)
              dddd is the function's descriptive name                  */


/* ********************************************************************
   kl_gecca : approximate eccentric anomaly from mean anomaly and time
   ******************************************************************** */
   Angle
kl_gecca(theorbit, from_epoch)
   OrbitType         *theorbit;
   Time              from_epoch;
{
/*  Newton iteration method for solving Kepler's Equation :
                En+1 = En + (M - Mn) / (dM/dE)
    Solution from "Fundamentals of Astrodynamics" (Pp.220-222)
             by Bate, Mueller and White                                */
int                  iter;     /* number of iterations accompished     */
Angle                en;       /* approx eccentric anomaly             */
Angle                mn;       /* approx mean anomaly                  */
Angle                m;        /* computed (exact) mean anomaly        */
```

```
   m  = theorbit->mean_anom +
        gl_gmean(theorbit->semimajor) * (from_epoch - theorbit->epoch);
                                /* M = M0 + n (t - t0)                    */
   if (m == 0.0)                /* by definition                          */
      en = theorbit->eccentric_anom;
   else
   {
      en = PI;                  /* intial guess normally converges        */
      mn = en;                  /* mean anomaly init assume as same       */
      iter = 0;
      while ( (abs(m - mn)>GL_EPSILON) && (iter<10) )
      {
         iter = iter + 1;
         en = en + (m - mn) / (1.0 - (theorbit->eccentricity * cos(en)));
         mn = en - (theorbit->eccentricity * sin(en) );
      }
   }

   theorbit->eccentric_anom = en;
   return( theorbit->eccentric_anom );
}   /* ************ function approximate eccentric anomaly ********** */



/* ************************************************************************
   kl_geccp : get eccentric anomaly from mean anomaly and time
              with first order aspherical gravitational perturbations
   ************************************************************************ */
   Angle
kl_geccp(theorbit, from_epoch)
   OrbitType         *theorbit;
   Time              from_epoch;
{
/*  Use kl_gecca after updating mean anomaly (M), long of ascending
    node (ASN) and argument of perigee (ARGP) by the formulas:
          M    = M    + M'    (t-t0)
          ASN  = ASN  + ASN'  (t-t0)
          ARGP = ARGP + ARGP' (t-t0)
    (from Smith pp. 94 )                                                  */

Angle               m;          /* new mean anomaly (M)                 */
Angle               asn;        /* long of ascending node (ASN)         */
Angle               argp;       /* argument of perigee (ARGP)           */
Angle               m_chg;      /* rate of change of mean anomaly       */
Angle               asn_chg;    /* rate of change of long of ascend node*/
Angle               argp_chg;   /* rate of change of argument of perigee*/
Dist                a;          /* semi-major axis                      */
Dist                psqr;       /* semi parameter squared               */
Angle               i;          /* inclination                          */
```

```
Real                sinsqr;   /* sin i * sin i                          */
Real                e;        /* eccentricity                           */

   a = gl_gorba(*theorbit);
   e = gl_gorbe(*thecrbit);
   i = cv_gangr(gl_gorbi(*theorbit));
   sinsqr = sin(i) * sin(i);

   psqr = gl_gsemi(a, e);
   psqr = psqr * psqr;

   m_chg = gl_gmean(a) *
           (1.0 + 1.5 * gl_primary.J2 * sqrt(1-e*e) * (1-1.5*sinsqr) / psqr);

   asn_chg = -(1.5 * gl_primary.J2 * cos(i) / psqr) * m_chg;

   argp_chg =  -(1.5 * gl_primary.J2 * (2.0 - 2.5*sinsqr) / psqr) * m_chg;


                              /* Update the orbital elements in place */
   theorbit->mean_anom =
      theorbit->mean_anom + m_chg * (from_epoch - theorbit->epoch);
                              /* M = M0 + M' (t - t0)                   */
   theorbit->long_of_asc_node =
      theorbit->long_of_asc_node + asn_chg * (from_epoch - theorbit->epoch);
                              /* ASN = ASN  + ASN' (t - t0)             */
   theorbit->arg_of_perigee =
      theorbit->arg_of_perigee + argp_chg * (from_epoch - theorbit->epoch);
                              /* ARGP = ARGP + ARGP' (t - t0)           */

   return( kl_gecca(*theorbit, from_epoch) );
}   /* ************ function perturbed eccentric anomaly **********   */
```

```
/*********************************************************************

          FILENAME             :  timeLib.c (tl_)
          DESCRIPTION          :  Time Calculation StdLib source file
          ENVIRONMENT          :  Macintosh SE 1Mb RAM
                                  LightSpeed™ C v2.15
          AUTHOR               :  Captain Kenneth L. BEUTEL USMC
          ADVISORS             :  Prof. Dan Davis
                                  Prof. Dan Boger
                                  Naval Postgraduate School, Monterey CA
          REMARKS              :  none
          VERSION              :  0.9 (3/6/88)

          CHANGES              :  3/6/88 Formatted for MacWrite conversion

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

#include <stdio.h>
#include <storage.h>
#include <math.h>
#include <strings.h>

/* include the StdLib header file's typedefs and global variables     */
#include "StdLib.h"


/* Because this is a library no main procedure is allowed or required
   all function names follow the standard convention :
          tl_adddd - where tl is the library name (time library)
          a          is the action that the function performs (such as
                     initialize, get, set, and query)
          dddd is the function's descriptive name                     */


/* ********************************************************************
   Special Notes : Time is considered to be in sidereal radian units
                   unless explicitly specified.

                   Angular measures are abstracted to remove the
                   requirement for unit specifications.
   ********************************************************** */

/* year table is a tabular representation of dates and times
   (ex[pressed as a radian angle) used to approximate Greenwich sidereal
   time by function tl_gknow().                                       */

#define   TL_TABLE_SIZE       21
static    Real                tl_year_table[TL_TABLE_SIZE][2] =
   {
      1968.0, 1.74046   ,  1969.0, 1.75350   ,  1970.0, 1.74933,
```

```
      1971.0, 1.74517   ,  1972.0, 1.74106   ,  1973.0, 1.75411,
      1974.0, 1.74995   ,  1975.0, 1.74577   ,  1976.0, 1.74583,
      1977.0, 1.75461   ,  1978.0, 1.75042   ,  1979.0, 1.74624,
      1980.0, 1.74208   ,  1981.0, 1.75511   ,  1982.0, 1.75088,
      1983.0, 6.67192   ,  1984.0, 1.74262   ,  1985.0, 1.74732,
      1986.0, 1.75149   ,  1987.0, 1.74733   ,  1988.0, 1.74316
   };



/* ********************************************************************
   tl_gzone : get time zone number (hours from GMT) at observer's longitude
   ******************************************************************** */
   Real
tl_gzone(longitude)
   Angle             longitude;
{                              /* do the equivalent of modulus         */
Real                 x;

   x = cv_gangd(longitude);
   for ( ; (x > 360.0) ; x = x - 360.0);
   for ( ; (x <   0.0) ; x = x + 360.0);

   return( floor( x / 15.0) );
}   /* ************* function get time zone ************************ */


/* ********************************************************************
   tl_gznam : get time zone name (std abbrev) for given time zone number
   ******************************************************************** */
   char *
tl_gznam(number)
   Real              number;
{
int                  thezone;

   thezone = number;/* coerce to integer type                 */
   if ( (thezone < 0) || (thezone > 24) )
      return("INV.");          /* INValid                              */

   switch (thezone)
   {                           /* No need to break because of return   */
   case 24:
   case 0:
      return("GMT ");          /* Next 8 std time zones from the U.S.
                                  Uniform Time Act of 1966 :           */
   case 4:
      return("AST ");
```

```
   case 5:
      return("EST ");
   case 6:
      return("CST ");
   case 7:
      return("MST ");
   case 8:
      return("PST ");
   case 9:
      return("YST ");
   case 10:
      return("AHST");
   case 11:
      return("BST ");
   default:                    /* No source for name conventions       */
      return("????");
   }
}   /* ************* function get time zone name ******************** */


/* ************************************************************************
   tl_gknow : get known Greenwich sidereal time (rad) from tabulated time
   ************************************************************************ */
   Time
tl_gknow(known_date)
   Real              known_date;
{
Time                 known_time;
int                  i;
Real                 dummy, month, day, known_year;

   tl_gmdyr(known_date, &month, &day, &known_year);

   if ((month != 1.0) || (day != 1.0))
      return( (Time) -1.0 );  /* only Jan 1st dates are tabulated     */

   known_time = -1.0;         /* assume time requested not in table   */
   for (i=0; i<TL_TABLE_SIZE; i=i+1)
   {
       if (tl_year_table[i][0] == known_year)
          known_time = (Time) tl_year_table[i][1];
   }                          /* get corresponding time for that yr   */

   return( known_time );
}   /* ********* function get known Greenwich sidereal time ********* */


/* ************************************************************************
   tl_ggren : get Greenwich sidereal time (approx) in radians
```

```
   ************************************************************** */
   Angle
tl_ggren(datetime)
   DateTime          datetime;
{
Time                 known_time, approx_time;
Real                 dummy, start_year, known_date, num_rotations;

   tl_gmdyr(datetime.date, &dummy, &dummy, &start_year);
   if (start_year < 1955)
      return( -1.0 );          /* no data on record before 1955        */

   known_date = tl_gjuld(1.0,1.0,start_year);
   for ( ; known_date>= datetime.date ; )
   {                           /* scan for 1st rec less than desired   */
      start_year = start_year - 1.0;
      known_date = tl_gjuld(1.0, 1.0, start_year);
   }

   known_time = tl_gknow(known_date);
                               /* get known time for that date         */

   num_rotations = datetime.date - tl_gjuld(1.0,1.0,start_year)
                 + (datetime.time / 24.0);
                               /* per day                              */

   approx_time = known_time +
               (Time) (gl_primary.ang_rot * 24.0 * num_rotations);
   return( approx_time );
} /* *************** function get Greenwich sidereal time *********** */



/* ********************************************************************
   tl_glstm : get local sidereal time at observers longitude.
   ************************************************************** */
   Angle
tl_glstm(longitude, datetime)
   Angle             longitude;
   DateTime          datetime;
{
Real                 radians_east;

   radians_east = cv_gangr(longitude);
   return( tl_ggren(datetime) + radians_east  );
}   /* ************** function get local sidereal time ************** */



/* ********************************************************************
```

```
   tl_gjuld : get julian date (in whole days).
   ************************************************************** */
   Real
tl_gjuld(month, day, year)
   Real              month, day, year;
{/* Range of data is : 1 <= month <= 12, 1<= day <= 31, 1700 <= year
    This routine is from pp. 19 of "119 Practical Programs for the
    TRS-80 Pocket Computer" by John Clark Craig, TAB Books (1982).    */
Real      julian;              /* declare result                      */
Real      n, z, e, w, x;       /*    and some temps          */


   julian = floor(365.2422*year + 30.44*(month-1.0) + day + 1.0);
   n = month - 2.0 + 12.0 * (month<3.0);
   z = year - (month<3.0);
   e = floor(z/100.0);
   z = z - 100.0 * e;
   w = floor(2.61 * n - 0.2) + day + z + floor(z/4.0)
       + floor(e/4.0) - 2.0*e;
   w = w - 7.0 * floor(w/7.0);
   x = julian - 7.0 * floor(julian/7.0);
   julian = julian - x + w - 7.0 * (x<w) + 1721061.0;

   return (julian);
}   /* ************* function get julian date ********************** */


/* ************************************************************************
   tl_gmdyr : get month, day and year from julian date (in whole days).
   ************************************************************************ */
   void
tl_gmdyr(julian, month, day, year)
   Real              julian;
   Real              *month, *day, *year;
{
/*  This routine is from pp. 19-20 of "119 Practical Programs for the
    TRS-80 Pocket Computer" by John Clark Craig, TAB Books (1982).    */
Real                 savejulian;
                              /* declare result                      */

   savejulian = julian;
/* solve for year by making a guess and backing into the answer       */
   *year = floor( (julian-1721061.0) / 365.25 + 1.0);
   *month = 1.0;
   *day = 1.0;
   julian = tl_gjuld(*month, *day, *year);
   for ( ; julian>savejulian ; )
   {
      *year = *year - 1.0;
      julian = tl_gjuld(*month, *day, *year);
```

```
   }

/* now solve for month by making a guess and backing into the answer  */
   *month = floor( (savejulian-julian) / 29.0 + 1.0);
   julian = tl_gjuld(*month, *day, *year);
   for ( ; julian>savejulian ; )
   {
      *month = *month - 1.0;
      julian = tl_gjuld(*month, *day, *year);
   }

/* remainder is the number of days in the month                       */
   *day = savejulian - julian + 1.0;
}   /* ************* function get month, day, year ****************** */
```

APPENDIX C STANDARD LIBRARY DOCUMENTATION

This appendix documents the services available from the standard library in a specific format. The format used by this appendix is similar to the conventions used in the documentation of UNIX commands and system calls. Each function or group of related functions that is found in the standard library is described independently on a separate page. Various type styles are used in a formal way to differentiate the exact purpose that each item of text provides.

The type style formatting rules include:

- The use of bold face and capital letters for section headings (e.g., **NAME**)
- The use of bold face for function names when they are mentioned inside of a body of text (e.g., **gl_gangd()**)
- The use of italics to designate parameters to functions when they are mentioned inside of a body of text (e.g., *dateinput*)

The format of each documentation page is as follows:

- A function header which lists the function name and the library it is located in. The function name is followed by ellipses if the documentation page is for more than one function.
- A name section consisting of a section header and the name of all functions documented by this page
- A syntax section that lists each function and its associated arguments
- A description section that describes the duty of each function
- An optional section that describes the return value, if one exists.
- An optional section that describes any known difficiencies or input cases not handled by the documented functions
- An optional "see also" section that cross references the user to other related functions

The standard library defines several abstract data types which are used repeatedly by these various routines. These values are declared in the header file "StdLib.h" of the standard library source code. The principle data structures are:

- Angle - used to define an abstract storage representation for angle measures
- Dist - used to define an abstract storage representation for length measures
- Time - used to define an abstract storage representation for time measures
- Real - used to represent other floating point numbers not in abstracted form
- DateTime - used to represent the aggregate of specific julian date and time of day
- OrbitType - used to represent the aggregate of elements comprising an orbit
- PQWCoord - used to represent the aggregate of 3 PQW position elements
- IJKCoord - used to represent the aggregate of 3 IJK position elements
- GEOCoord - used to represent the aggregate of 3 GEO position elements
- RADCoord - used to represent the aggregate of 3 RA-Decl position elements
- SEZCoord - used to represent the aggregate of 3 SEZ position elements
- PrimaryType - used to represent the aggregate of elements comprising the primary

NAME

gl_idflt();

SYNTAX

```
#include "StdLib.h"
void    gl_idflt();
```

DESCRIPTION

this function initializes the entire standard library package with the earth as the default primary body.

INPUT RESTRICTIONS

The use of this function is required before any other standard library functions are called or anomalous results may occur.

NAME

gl_gpnam ();

SYNTAX

```
#include "StdLib.h"
char      *gl_gpnam();
```

DESCRIPTION

this function retrieves the string value that names the primary body currently being used.

RETURN VALUE

a pointer to a previously allocated C string variable.

INPUT RESTRICTIONS

After initialization, the default value returned by this function is "Earth".

SEE ALSO

gl_idflt();

NAME

gl_sorbt();

SYNTAX

```
#include "StdLib.h"
void    gl_sorbt(char *name, Real a, Real e, Angle i,
            Angle mean_anom, Angle peri, Angle asn,
            Time epoch, Real date,  OrbitType *orbrec );
```

DESCRIPTION

this function takes a standard set of orbit parameters and stores them in the *orbrec* structure in an internal format. This function should be used to establish values for all orbit records used by the standard library.

INPUT RESTRICTIONS

The variables using the abstract data types Angle and Real should be set by Conversion Library functions.

SEE ALSO

cv_sangd();
cv_sangr();
cv_cdisk();
cv_ckdis();
cv_ckspd();
cv_cspdk();

NAME

gl_gorbn();
gl_gorba();
gl_gorbe();
gl_gorbi();
gl_gorbm();
gl_gorbp();
gl_gorbl();
gl_gorbt();

SYNTAX

```
#include "StdLib.h"
char    *gl_gorbn (OrbitType orbit);
Dist    *gl_gorba (OrbitType orbit);
Real    *gl_gorbe (OrbitType orbit);
Angle   *gl_gorbi (OrbitType orbit);
Angle   *gl_gorbm (OrbitType orbit);
Angle   *gl_gorbp (OrbitType orbit);
Angle   *gl_gorbl (OrbitType orbit);
Time    *gl_gorbt (OrbitType orbit);
```

DESCRIPTION

these functions return a single orbital element from the abstract OrbitType variable *orbit*. The values returned are the orbit name (**gl_gorbn**), semimajor axis (**gl_gorba**), eccentricity (**gl_gorbe**), inclination (**gl_gorbi**), mean anomaly (**gl_gorbm**), argument of perigee (**gl_gorbp**), longitude of the ascending node (**gl_gorbl**), and epoch time (**gl_gorbt**), respectively.

RETURN VALUE

Each of these functions returns a single abstract data value. The value returned may then be manipulated via the Conversion Library to obtain the desired system of units.

SEE ALSO

cv_gangd();
cv_gangr();
cv_cdisk();
cv_ckdis();
cv_ckspd();
cv_cspdk();

NAME

gl_grada();
gl_gradp();

SYNTAX

```
#include "StdLib.h"
Dist    gl_grada(Dist a, Real e);
Dist    gl_gradp(Dist a, Real e);
```

DESCRIPTION

these functions calculate the radius of apogee (**gl_grada**) and radius of perigee (**gl_gradp**) from an orbit's semimajor axis (*a*) and eccentricity (*e*).

RETURN VALUE

a distance measure that is in the same system of units as the semimajor axis,*a*.

NAME

gl_gradi();

SYNTAX

```
#include "StdLib.h"
Dist      gl_gradi(Dist p, Real e, Angle nu);
```

DESCRIPTION

these functions calculate the radial distance to a satellite from the orbit's semiparameter (*p*) eccentricity (*e*), and true anomaly angle (*nu*).

RETURN VALUE

a distance measure that is in the same system of units as the semiparameter,*p*.

SEE ALSO

gl_gsemi();

NAME

gl_true();

SYNTAX

```
#include "StdLib.h"
Angle    gl_true(Dist x, Dist y, Dist p, Real e);
```

DESCRIPTION

this function obtains the true anomaly angle to a satellite from its in-plane x and y coordinates (x,y), semiparameter (p), and eccentricity (e).

RETURN VALUE

an abstract angular measure that should be accessed via the Conversion Library routines.

INPUT RESTRICTIONS

the distance measures must all be in the same system of units.

SEE ALSO

cv_gangd();
cv_gangr();

NAME

gl_gxpos();
gl_gypos();

SYNTAX

```
#include "StdLib.h"
Dist    gl_gxpos(Dist p, Real e, Angle nu);
Dist    gl_gypos(Dist p, Real e, Angle nu);
```

DESCRIPTION

these functions obtain the in-plane x and y coordinates of the given orbit from the semiparameter (*p*), eccentricity (*e*) and true anomaly (*nu*).

RETURN VALUE

a distance measure that is in the same system of units as the semiparameter,*p*.

NAME

gl_gsemi();

SYNTAX

```
#include "StdLib.h"
Dist      gl_gsemi(Dist a, Real e);
```

DESCRIPTION

this function calculates a value for the semiparameter from the semi-major axis (*a*) and eccentricity (*e*).

RETURN VALUE

a distance measure that is in the same system of units as the semi-major axis (*a*).

NAME

gl_gmean();

SYNTAX

```
#include "StdLib.h"
Real    gl_gmean(Dist a);
```

DESCRIPTION

this function calculates a value of the mean motion constant from the semi-major axis, *a*.

RETURN VALUE

a distance measure that is in the same system of units as the semi-major axis (*a*) and the gravitational parameter that the system is defined for (initialized for kilometers).

NAME

gl_gperd();

SYNTAX

```
#include "StdLib.h"
Time    gl_gperd(Real n);
```

DESCRIPTION

this function calculates the period of a satellite from the mean motion constant (n).

RETURN VALUE

a time measure that is in the same system of units as the gravitational parameter that the system is defined for (initialized for seconds).

SEE ALSO

gl_gmean();

NAME

gl_gvelo();
gl_gvesc();

SYNTAX

```
#include "StdLib.h"
Real    gl_gvelo(Dist r, Dist a);
Real    gl_gvesc(Dist r);
```

DESCRIPTION

this function calculates the orbital velocity and the escape velocity of a satellite with semi-major axis (*a*) and radial distance (*r*) from the primary.

RETURN VALUE

a speed measure that is in the same system of units as the gravitational parameter that the system is defined for (initialized for km/seconds).

SEE ALSO

gl_gradi();

NAME

gl_gfltp();

SYNTAX

```
#include "StdLib.h"
Angle    gl_gfltp(Real h, Dist r, Real v);
```

DESCRIPTION

this function calculates the flight path angle of a satellite at a radial distance (r) from the primary using velocity (v) and angular momentum (h).

RETURN VALUE

an abstract angular measure that should be accessed via the Conversion Library routines.

INPUT RESTRICTIONS

velocity must be specified in the same system of units as the gravitational parameter that the system is defined for (initialized for km/seconds).

SEE ALSO

gl_gangm();
gl_gradi();
gl_gvelo();

NAME

gl_glosd();
gl_ggswi();

SYNTAX

```
#include "StdLib.h"
Dist    gl_glosd(Dist r, Angle gamma);
Dist    gl_ggswi(Dist r);
```

DESCRIPTION

these functions calculate the line of sight distance to the horizon and the geometric swath width at a radial distance (*r*).

RETURN VALUE

an arbitrary distance measure that can be modified via the Conversion Library routines.

INPUT RESTRICTIONS

the angle *gamma* should be an acute angle measured above the local horizontal.

SEE ALSO

gl_gradi();

NAME

gl_gangm();

SYNTAX

```
#include "StdLib.h"
Real    gl_gangm(Dist a, Real e);
```

DESCRIPTION

this function calculates the magnitude of the angular momentum of a satellite at a radial distance (*r*) from the primary.

SEE ALSO

gl_gradi();

NAME

gl_gspen();

SYNTAX

```
#include "StdLib.h"
Real      gl_gspen(Real v, Dist r);
```

DESCRIPTION

this function calculates the specific energy of a satellite moving with velocity (*v*) at a radial distance (*r*) from the primary.

INPUT RESTRICTIONS

speed and distance measures must be in the same system of units as the gravitational parameter that the system is defined for (initialized for km/seconds).

SEE ALSO

gl_gvelo();
gl_gradi();

NAME

tl_gzone();
tl_gznam();

SYNTAX

```
#include "StdLib.h"
Real    tl_gzone(Angle longitude);
char    *tl_gznam(Real zonenumber);
```

DESCRIPTION

the function **tl_gzone** performs a coarse time zone calculation for the given longitude.
when given the *zonenumber* the function **tl_gznam** will determine the standard abbreviation for that time zone.

RETURN VALUE

the function **tl_gzone**() returns the number of the time zones (hours) west of the Greenwich meridian. This function does not correct for time zone deviations from true longitude or daylight savings time.
the function **tl_gznam**() returns a pointer to a four character string containing the time zone abbreviation.

INPUT RESTRICTIONS

the input to **tl_gzone**() is an abstract angular measure that should be accessed via the Conversion Library routines.
tl_gznam() will accept any numeric value but will only produce valid results for real numbers representing the integers 1-24.

NAME

tl_gknow();

SYNTAX

```
#include "StdLib.h"
Time    tl_gknow(Real known_date);
```

DESCRIPTION

for dates that exist in an internal table this function determines the conversion from universal time to sidereal time.

RETURN VALUE

for valid dates, the difference between the time systems is returned, -1 otherwise.

INPUT RESTRICTIONS

any julian date in real format is accepted as input but the table may not contain results for that specific date.

NAME

tl_ggren();
tl_glstm();

SYNTAX

```
#include "StdLib.h"
Angle   tl_ggren(DateTime datetime);
Angle   tl_glstm(Angle longitude, DateTime datetime);
```

DESCRIPTION

the function **tl_ggren** approximates Greenwich Sidereal Time for a given *datetime* record.
the function **tl_glstm** approximates the local sidereal time for an observer at *longitude* east of the Greenwich Meridian.

RETURN VALUE

an abstract angular measure that should be accessed via the Conversion Library routines.

INPUT RESTRICTIONS

the *datetime* structure should contain valid values for both time of day and julian date.

SEE ALSO

tl_gknow();

NAME

tl_gjuld();
tl_gmdyr();

SYNTAX

```
#include "StdLib.h"
Real    tl_gjuld(Real month, Real day, Real year);
void    tl_gjuld(Real julian, Real *month, Real *day,
            Real *year);
```

DESCRIPTION

a pair of complementary functions that convert between julian date and month, day year date formats.

RETURN VALUE

the number of julian days in real format is returned by **tl_gjuld**. The function **tl_gjuld** uses pass by reference to supply real number representations for the month, day, and year.

INPUT RESTRICTIONS

if invalid dates are specified in either format the results are unpredictable .

NAME

cv_sangd();
cv_sangr();

SYNTAX

```
#include "StdLib.h"
Angle    cv_sangd(Angle degrees);
Angle    cv_sangr(Angle radians);
```

DESCRIPTION

these functions store angle values, specified in *degrees* or *radians*, in an internal representation.

RETURN VALUE

an abstract data representation of the angle specified.

SEE ALSO

cv_gangd();
cv_gangr();

NAME

cv_gangd();
cv_gangr();

SYNTAX

```
#include "StdLib.h"
Angle    cv_gangd(Angle value);
Angle    cv_gangr(Angle value);
```

DESCRIPTION

these functions return angular representation for angles in degrees (**cv_gangd**) or radians (**cv_gangr**) from an internal *value*.

RETURN VALUE

an angle measured in degrees and radians, respectively.

SEE ALSO

cv_sangd();
cv_sangr();

NAME

cv_gsolt();
cv_gsidt();

SYNTAX

```
#include "StdLib.h"
Time    cv_gsolt(Time value);
Time    cv_gsidt(Time value);
```

DESCRIPTION

this pair of complementary functions convert between sideral (**cv_gsolt**) and solar (**cv_gsidt**) formats.

INPUT RESTRICTIONS

the time values must be in a compatible system of units.

NAME

cv_cdisk();
cv_ckdis();
cv_cspdk();
cv_ckspd();
cv_chtim();
cv_ctimh();

SYNTAX

```
#include "StdLib.h"
Dist    cv_cdisk(Dist canon_distance);
Dist    cv_ckdis(Dist km_distance);
Real    cv_cspdk(Real canon_speed);
Real    cv_ckspd(Real km_sec);
Time    cv_chtim(Time canon_time);
Time    cv_gtimh(Time sec_time);
```

DESCRIPTION

these three pairs of complementary functions convert between the metric system of units and canonical units.
The function **cv_cdisk** converts from *canon_distance* to kilometers.
The function **cv_ckdis** converts from *km_distance* to canonical distance (radii of the primary).
The function **cv_spdk** converts from *canon_speed* to kilometers/sec.
The function **cv_ckspd** converts from *km_sec* to canonical speed (radii per herg).
The function **cv_chtim** converts from *canon_time* (hergs) to kilometers.
The function **cv_ctimh** converts from *sec_time* (seconds) to canonical time (hergs).

INPUT RESTRICTIONS

the values must be in the specified system of units.

NAME

cs_gpqwc();
cs_gijkc();
cs_gpqwk();
cs_gijks();
cs_gijkr();
cs_gradc();
cs_ggeoc();
cs_gijkg();

SYNTAX

```
#include "StdLib.h"
void    cs_gpqwc(OrbitData orbit,
          PQWCoord *pqw_pos);
void    cs_gijkc(OrbitData orbit, IJKCoord *ijk_pos);
void    cs_gpqwk(OrbitData orbit, IJKCoord ijk_pos,
        PQWCoord *pqw_pos);
void    cs_gijks(OrbitData orbit, SEZCoord sez_pos,
        Angle lat, Angle lon, IJKCoord *ijk_pos);
void    cs_gijkr(OrbitData orbit, RADCoord rad_pos,
        IJKCoord *ijk_pos);
void    cs_radc(OrbitData orbit, IJKCoord ijk_pos,
        RADCoord *rad_pos);
void    cs_ggeoc(OrbitData orbit,
        GEOCoord *geo_pos);
void    cs_gijkg(OrbitData orbit, GEOCoord geo_pos,
        IJKCoord *ijk_pos);
```

DESCRIPTION

these functions calculate the position of a satellite from *theorbit* in one of several different coordinate systems.

RETURN VALUE

each function uses pass by reference to supply an abstract data representation of position from *theorbit*.

NAME

kl_gecca();
kl_geccp();

SYNTAX

```
#include "StdLib.h"
Angle   kl_gecca(OrbitType theorbit, Time frm_epoch);
Angle   kl_geccp(OrbitType theorbit, Time frm_epoch);
```

DESCRIPTION

this function uses an iterative approach to solving the Kepler Problem of determining the location of a satellite. The satellite is initially located at a position provided by *theorbit* and a new position is requested at *from_epoch* units of time later.

INPUT RESTRICTIONS

the *from_epoch* time value must be in seconds.

APPENDIX D

MACORBITS SOURCE LISTING

```
#ifndef  _MacOrbits_
#define  _MacOrbits_

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

          FILENAME            :  MacOrbits.h
          DESCRIPTION         :  header file for MacOrbits project
          ENVIRONMENT         :  Macintosh SE 1Mb
                                 LightSpeed™ C v2.15
          AUTHOR              :  Captain Kenneth L. BEUTEL USMC
                                 (Some portions Copyright Think Technologies)
          ADVISORS            :  Prof. Dan Davis
                                 Prof. Dan Boger
                                 Naval Postgraduate School, Monterey CA
          REMARKS             :  Contains globally used constants/variables
          VERSION             :  0.9 (3/6/88)

          CHANGES             :  3/6/88 Formatted for MacWrite conversion
********************************************************************** */


/* resource ID's of windows, alerts and dialogs                        */
#define   windowID            128
#define   GenericAlertID      256
#define   DirtyFileID         257
                              /* advise saving changes before quit     */
#define   AboutAlertID        258
                              /* about Orbits... alert                 */
#define   NewOrbitID 512
                              /* define a new orbit... dialog          */
#define   SetUnitsID 513
                              /* measurement units... dialog           */
#define   TimeStepID 514
                              /* set time step... dialog               */
#define   TraceOrbitID        515
                              /* trace orbit path... dialog            */
#define   PlotDurationID      516
                              /* position obsever... dialog            */
#define   MapID               600
                              /* the map background PICT               */
#define   GlobeID             601
```

```
                        /* Base address for 3 Globe Picts          */

/* resource IDs of menus                                          */
#define   appleID           128
#define   fileID            129
#define   editID            130
#define   orbitsID          131
#define   plotID            132
#define   windID            133
#define   specialID         134

/* Menu indices  ***** NOTE : these correspond to xxxID - appleID  */
#define   appleM            0
#define   fileM             1
#define   editM             2
#define   orbitM            3
#define   plotM             4
#define   windowM           5
#define   specialM          6

/* File Menu items (4,8,11 are dimmed)                            */
#define   fmNew             1
#define   fmOpen            2
#define   fmClose           3
#define   fmSave            5
#define   fmSaveAs          6
#define   fmRevert          7
#define   fmPageSetUp       9
#define   fmPrint           10
#define   fmXFER            12
#define   fmQuit            13


/* Edit menu command indices                                      */
#define   undoCommand       1
#define   cutCommand 3
#define   copyCommand       4
#define   pasteCommand      5
#define   clearCommand      6


/* Plot Menu items (3 is dimmed)                                  */
#define   plONE             1
#define   plCONT            2
#define   plRESET           4
#define   plSTOP            6


/* DirtyFile alert Dialog button numbers                          */
#define   dlgSave           1
```

```
#define   dlgDiscard 2
#define   dlgCancel         3


                            /* Other Important Definitions            */
#define   SBarWidth         15

                            /* Orbital Element display box size       */
#define   BOX_V             16
#define   BOX_H             68

#define   INFRONT           -1L
#define   NIL               0L
#define   OFF               0
#define   ON                1
                            /* 5 total windows available to user      */
#define   MAXWINDOWS 5
#define   MAXGLOBES  3
                            /* top(V) left(H) and MaxRight            */
#define   WINDOW_V          40
#define   WINDOW_H          30
#define   WH_MAX            455


                            /* Topleft corner of map                  */
#define   MAP_TOP           80
#define   MAP_LEFT          40

/* Number of degrees of latitude and longitude per screen pixel in Map*/
#define   PIX_LAT           1.143
#define   PIX_LON           1.153

/* Potential coordinate systems that window can be drawn in          */
#define   IJK_COORDS 0
#define   PQW_COORDS 1
#define   GEO_COORDS 2

/* origin position for PQW coord windows                             */
#define   PQW_X             90
#define   PQW_Y             200

typedef  char Str30[30];    /* a shorter string for storage          */

typedef struct
   {
      int           slotnum;
                            /* current slot window occupies          */
      Boolean       dirty;  /* has data for orbit been changed?      */
      Boolean       newfile;
                            /* source of data is not from disk file  */
```

```
    Boolean       textonly;
                        /* display data in text (no pics)          */
    int           coordinates;
                        /* current coord system for window         */
    Str255        test;
    int           lastview;
                        /* current view of spinning globe          */
    IJKCoord      ijk;    /* current IJK position                   */
    PQWCoord      pqw;    /* current PQW position                   */
    GEOCoord      geo;    /* current GEO position                   */
    IJKCoord      last_ijk;          /* last IJK position          */
    PQWCoord      last_pqw;          /* last PQW position          */
    GEOCoord      last_geo;          /* last GEO position          */
    OrbitType     orbitdata;/* Values for the orbit                */
  } OrbitInfo;

typedef struct
  {
    Boolean       changes;/* changes to the Menu items ?           */
    int           bar_state;/* Menu bar state   0-off 1-on         */
    Boolean       showmap;/* display/hide the map                  */
    int           std_units;/* metric(0), canonical(1)             */
    double        time_comp;/* time compression factor (x:1)       */
    long int      elapsed_time;
                        /* numTicks ago when last plot was done    */
    long int      time;   /* Wall clock time in seconds            */
    long int      stop_time;/* stop plotting action (min)          */
    Boolean       plotting;/* currently computing orbits?          */
    int           plot_duration;
                        /* plot plONE or plCONT                    */
    long int      draw_method;
                        /* plot (0)dots (1)lines (2)both           */
    Boolean       first_plot;
                        /* first time in curr plot seq             */
    double        draw_incr;/* increment plot every x minutes      */
    Boolean       showaxes;/* show axes when plotting             */
    double        obs_lat;/* latitude of observer                 */
    double        obs_lon;/* longitude of observer                */
  } Preferences;

#endif
```

```
/*********************************************************************

          FILENAME            :  MacOrbits.proto.c
          DESCRIPTION         :  prototypes for functions used in MacOrbits
          ENVIRONMENT         :  Macintosh SE 1Mb
                                 LightSpeed™ C v2.15
          AUTHOR              :  Captain Kenneth L. BEUTEL USMC
          ADVISORS            :  Prof. Dan Davis
                                 Prof. Dan Boger
                                 Naval Postgraduate School, Monterey CA
          REMARKS             :  none
          VERSION             :  0.9 (3/6/88)

          CHANGES             :  3/6/88 Formatted for MacWrite conversion
*********************************************************************** */

#ifndef   _MacOrbitsProto_
#define   _MacOrbitsProto_

                              /* found in MacOrbits.c                    */
pascal void MO_ResumeProc();
int       MO_MainEvent();
void      MO_SetUpMenus();
void      MO_MenuClick( long mResult );
void      MO_About();
void      MO_MaintainMenus();
void      MO_SetUpCursors();
void      MO_Init_Preferences(Preferences *prefer);
void      MO_Init_Globes();

                              /* found in MacOrbits.fm.c                 */
int       MO_File(int item, char *theFileName, int *theVRefNum,
                  WindowPtr myWindow);
int       MO_SaveAs(Str255 *name, int *vRefnum, WindowPtr myWindow );
int       MO_SaveFile(char *name, int vRefNum, WindowPtr myWindow);
int       MO_Advise(Str255 *s);
int       MO_NewFile(Str255 *name, int *vRefnum);
int       MO_OldFile(Str255 *name,int *vRefnum);
int       MO_OldApps(Str255 *name,int *vRefnum);
int       MO_CreateFile(Str255 *name, int *vRefnum, int *theRef);
int       MO_WriteFile(int refNum, WindowPtr myWindow);
int       MO_ReadFile(int   refNum, WindowPtr myWindow);
int       MO_ReadLine(char *buffer, int refNum);
void      MO_fmNew(WindowPtr *myWindow, int slotnum);

                              /* found in MacOrbits.map.c                */
void      MO_CreateMap(BitMap           *bm,WindowPtr *mapWindow);
void      MO_DeleteMap(BitMap           *bm,WindowPtr *mapWindow);
void      MO_DrawMap(BitMap bm, WindowPtr mapWindow);
int       MO_LattoPixel(double lat);
```

```
int       MO_LontoPixel(double lon);

                          /* found in MacOrbits.menu.c                */
void      MO_OrbitsMenu(int theItem, WindowPtr theWind);
void      MO_PlotMenu(int theItem);
void      MO_SetPlotDuration();
void      MO_WindowMenu(int theItem);
void      MO_SpecialMenu(int theItem);
void      MO_SetUnits();
void      MO_TimeStep();
void      MO_OrbitTrace();

                          /* found in MacOrbits.pl.c                  */
void      MO_TextOnly(OrbitInfo orbitinfo);
void      MO_DrawElt(int row, int col, char *str);
void      MO_DrawALL(OrbitInfo orbitinfo, WindowPtr theWind);
void      MO_DrawIJK(OrbitInfo orbitinfo, WindowPtr theWind);
void      MO_DrawPQW(OrbitInfo orbitinfo, WindowPtr theWind);
void      MO_DrawGEO(OrbitInfo orbitinfo, WindowPtr theWind);
void      MO_MaintainPlot( );

                          /* found in MacOrbits.pr.c                  */
void      MO_CheckPrintHandle();
void      MO_PageSetUp();
void      MO_PrintOrbitText(WindowPtr theWind);
int       MO_HowMany();

                          /* found in MacOrbits.ut.c                  */
void      MO_Wait();
void      MO_pStrCopy( );   /* char *p1,  char *p2                    */
void      MO_pStrConcat( ); /* char *p1, char *p2, char *out          */
double    MO_pStr2Num(Str255 *str);
double    MO_pStr2Julian(Str255 *str);
void      MO_Generic();     /* Str255 s1,Str255 s2                    */
void      MO_OutlineButton(Rect r, DialogPtr theDialog);
void      MO_Pause(int x);
int       MO_trunc(double x);

                          /* found in MacOrbits.wm.c                  */
void      MO_CreateWindow(WindowPtr *theWind, int *slotnum);
Boolean   MO_isNewFilw(WindowPtr theWind);
int       MO_isDirty(WindowPtr theWind);
void      MO_SetDirty(WindowPtr theWind, Boolean thebit);
int       MO_ours(WindowPtr theWind);
Boolean   MO_AvailWind();
Boolean    MO_DuplicWind(Str255 newname);
void      MO_RemoveWindow(WindowPtr *theWind);
void      MO_HideWindow(WindowPtr theWind);
WindowPtr MO_FirstWindow(); /* First window used in storage list      */
void      MO_UpdateWindow(WindowPtr theWind);
```

```
void      MO_GrowWindow(WindowPtr theWind,Point p);
void      MO_ForceUpdate();
WindowPtr MO_NextWindow(WindowPtr current);

#endif
```

```
/*********************************************************************

          FILENAME            :  MacOrbits.c
          DESCRIPTION         :  main driver program contains main
                                 event loop
          ENVIRONMENT         :  Macintosh SE 1Mb
                                 LightSpeed™ C v2.15
          AUTHOR              :  Captain Kenneth L. BEUTEL USMC
          ADVISORS            :  Prof. Dan Davis
                                 Prof. Dan Boger
                                 Naval Postgraduate School, Monterey CA
          REMARKS             :  Contains Mac specific driver shell
                                 instructions
          VERSION             :  0.9 (3/6/88)

          CHANGES             :  3/6/88 Formatted for MacWrite conversion

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

#include "QuickDraw.h"
#include "MacTypes.h"
#include "FontMgr.h"
#include "WindowMgr.h"
#include "MenuMgr.h"
#include "TextEdit.h"
#include "DialogMgr.h"
#include "EventMgr.h"
#include "DeskMgr.h"
#include "FileMgr.h"
#include "ToolboxUtil.h"
#include "ControlMgr.h"
#include "stdio.h"

#include "LightSpeed Disk:Thesis C f:StdLib.h"

#include "MacOrbits.h"
#include "MacOrbits.proto.h"


Preferences          prefer; /* preferences for how things look        */
WindowPtr            myWindow;/* the current window of interest        */
WindowPtr            mapWindow = NIL;
                            /* the window that draws world map        */
Rect                 dragRect = { 0, 0, 1024, 1024 };
MenuHandle           myMenus[specialM + 1];
Cursor               watch;
BitMap               map_bm; /* where the original map is drawn        */
BitMap               globePics[MAXGLOBES];

Str255               theFileName;
```

```
static int          theVRefNum;


   /*********************************************************************
   MO_ResumeProc : allows user to "bail-out" to Finder if program bombs.
   ********************************************************************* */
   pascal void
MO_ResumeProc()
{
   ExitToShell();
}   /* ********************  MO_ResumeProc  ************************ */


   /*********************************************************************
   main : entry point for this application
   ********************************************************************* */
main()
{
   InitGraf(&thePort);      /* start up QuickDraw                    */
   InitFonts();             /*      ...  the Font Managaer           */
   TextFont(systemFont);    /* make the font ->Chicago               */
   TextSize(9);             /*           in 9 point size             */
   FlushEvents( everyEvent, 0 );
                            /* clear the Event Queue                 */
   InitWindows();           /* start up the Window Manager           */
   InitMenus();             /*           the Menu Manager            */
   TEInit();                /*           Text Edit                   */
   InitDialogs(&MO_ResumeProc);
                            /*           the Dialog Manager          */
   InitCursor();            /* reset the cursor to the arrow         */
   MoreMasters();           /* Create a couple extra blocks          */
   MoreMasters();           /*           of Master Pointers          */
   MoreMasters();
   MoreMasters();
   MaxApplZone();           /* create the application heap           */

   MO_SetUpCursors();       /* define all cursors used by prog       */
   MO_SetUpMenus();         /* create the program's menu bar         */
   MO_Init_Preferences(&prefer);
                            /* set up default session values         */
   MO_Init_Globes();/* load the globe pictures into mem              */

   gl_idflt();              /* Initialize the StdLibrary             */

   while ( MO_MainEvent() )
   {
      /* everything done in MainEvent                               */
   }
}   /* ********************* Main ********************************** */
```

```
   /*****************************************************************
   MO_MainEvent : polls main event loop
   ***************************************************************** */
   int
MO_MainEvent()
{
EventRecord          myEvent;
WindowPtr            whichWindow;
Rect                 r;
Boolean              saveprefer;
char                 chCode; /* Character code from evt msg            */


   MO_MaintainMenus();       /* Check if disable/enable any menu       */
   if (prefer.plotting)
      MO_MaintainPlot();
   SystemTask();             /* Give Time to DA's                      */

   if (GetNextEvent(everyEvent, &myEvent))
   {
      switch (myEvent.what)
      {
      case mouseDown:
         switch (FindWindow( myEvent.where, &whichWindow ))
         {
         case inDesk:
            SysBeep(3);
         break;
         case inGoAway:
            if (MO_ours(whichWindow))
               if (TrackGoAway( whichWindow, myEvent.where) )
                  MO_HideWindow(whichWindow);
         break;
         case inMenuBar:
            MO_MenuClick( MenuSelect(myEvent.where) );
         break;
         case inSysWindow:
            SystemClick( &myEvent, whichWindow );
         break;
         case inDrag:
            if (MO_ours(whichWindow))
               DragWindow( whichWindow, myEvent.where, &dragRect );
         break;
         case inGrow:
         if (MO_ours(whichWindow))
            MO_GrowWindow( whichWindow, myEvent.where );
         break;
         case inContent:
```

```
      if (whichWindow == mapWindow)
         ;              /* Do Nothing                                  */
      else if (MO_ours(whichWindow))
      {
         if (whichWindow != FrontWindow())
            SelectWindow(whichWindow);
         else
            ;           /* skipit-no valid mousedowns inContent        */
      }
   break;
   default:
      ;
   }                    /* end switch FindWindow                       */
break;
case keyDown:
case autoKey:
   chCode = myEvent.message & charCodeMask;
   if ((myEvent.modifiers & cmdKey) != 0)
     MO_MenuClick( MenuKey(chCode) );
   else
   {                    /* no response for non command key evt         */
   }
break;
case activateEvt:
   if ( ((WindowPtr) myEvent.message) == mapWindow)
      ;                 /* Do Nothing                                  */
   else if ( MO_ours((WindowPtr)myEvent.message))
   {
      SetPort((WindowPtr)myEvent.message);
      DrawGrowIcon(((WindowPtr)myEvent.message));
      if ( myEvent.modifiers & activeFlag )
      {                 /* window is becoming active                   */
                        /* Save plotting preferences                   */
         saveprefer = prefer.plotting;
         prefer.plotting = FALSE;
         MO_UpdateWindow((WindowPtr)myEvent.message);
         prefer.plotting = saveprefer;
                        /* And restore prefernces afterwards           */
      }
      else
      {                 /* window is becoming deactive                 */
      }
   }
   prefer.changes = TRUE;
break;
case updateEvt:
   if (MO_ours((WindowPtr)myEvent.message))
   {
      saveprefer = prefer.plotting;
      prefer.plotting = FALSE;
```

```
            MO_UpdateWindow((WindowPtr)myEvent.message);
            prefer.plotting = saveprefer;
         }
         else if ( ((WindowPtr) myEvent.message) == mapWindow)
            MO_DrawMap(map_bm, mapWindow);
      break;
      default: ;
      }                    /* end of case myEvent.what                    */
   }                       /* end if                                      */
   return(TRUE);
}   /* ********************* MO_MainEvent ************************** */



   /*********************************************************************
   MO_SetUpMenus :loads menu resources and sets up DA's under Apple menu
   ********************************************************************* */
   void
MO_SetUpMenus()
{
int                  i;

   myMenus[appleM] = GetMenu(appleID);
   AddResMenu( myMenus[appleM], 'DRVR' );

   for (i=fileID; i<=(appleID+specialM); i=i+1)
      myMenus[i-appleID] = GetMenu(i);

   for ( (i=appleM); (i<=specialM); i=i+1 )
      InsertMenu(myMenus[i], 0) ;

   DrawMenuBar();
}   /* ********************* MO_SetUpMenus ************************** */



   /*********************************************************************
   MO_MenuClick :processes the menu item or command key equiv.
   ********************************************************************* */
   void
MO_MenuClick( mResult )
   long              mResult;
{
int                  theItem;
Str255               DAname;
WindowPeek           wPtr;

   theItem = LoWord( mResult );
   switch (HiWord(mResult)) /* the Menu number                            */
   {
```

```
    case appleID:
       if (theItem == 1)     /* the about item                    */
          MO_About();
       else
       {
          GetItem(myMenus[appleM], theItem, &DAname);
          prefer.changes = TRUE;
                             /* Flag potential changes to menu    */
          OpenDeskAcc( &DAname );
       }
    break;
    case fileID:             /* set filename to top orbit window  */
          MO_File(theItem, (char *)theFileName,
                    &theVRefNum, FrontWindow());
    break;
    case editID:
       if (SystemEdit(theItem-1) == 0)
       {
          wPtr = (WindowPeek) FrontWindow();
          switch (theItem)
          {
          case cutCommand:
          break;
          case copyCommand:
          break;
          case pasteCommand:
          break;
          case clearCommand:
          break;
          default: ;
          }
       }
    break;
    case orbitsID:
       MO_OrbitsMenu(theItem, FrontWindow());
    break;
    case plotID:
       MO_PlotMenu(theItem);
    break;
    case windID:
       MO_WindowMenu(theItem);
    break;
    case specialID:
       MO_SpecialMenu(theItem);
    break;
    }
    HiliteMenu(0);
}   /* ********************* MO_MenuClick ************************** */
```

```
   /*****************************************************************
   MO_About : displays author and source statement
   ***************************************************************** */
   void
MO_About()
{
   Alert( AboutAlertID, 0L );
}   /* ******************** MO_About ******************************** */



   /*****************************************************************
   MO_MaintainMenus : enables/disables menu items based on program state
   ***************************************************************** */
   void
MO_MaintainMenus()
{
WindowPeek          theWindow;
OrbitInfo           *orbitinfo;

   if (!(prefer.changes))   /* No changes ... leave                    */
      return;

   prefer.changes = FALSE;  /* we are making the changes now           */


/* ************* Handle check items in menu bar here first           */
   if (prefer.showmap)      /* item 2 of Worldmenu is show/hide        */
   {
      CheckItem(myMenus[windowM], 1, TRUE);
      MO_CreateMap(&map_bm, &mapWindow);
   }
   else
   {
      CheckItem(myMenus[windowM], 1, FALSE);
      MO_DeleteMap(&map_bm, &mapWindow);
   }


   if (prefer.showaxes)     /* item 2 of Worldmenu is show/hide        */
   {
      CheckItem(myMenus[specialM], 4, TRUE);
   }
   else
   {
      CheckItem(myMenus[specialM], 4, FALSE);
   }

   theWindow = (WindowPeek) FrontWindow();
```

```
                              /* If it is an orbit window                */
   if (MO_ours( (WindowPtr) theWindow ))
   {                          /* clear all old checkmarks                */
      CheckItem(myMenus[orbitM], 1, FALSE);
      CheckItem(myMenus[orbitM], 2, FALSE);
      CheckItem(myMenus[orbitM], 4, FALSE);
      CheckItem(myMenus[orbitM], 5, FALSE);
      CheckItem(myMenus[orbitM], 6, FALSE);

      orbitinfo = (OrbitInfo *) (theWindow->refCon);
      if ( orbitinfo->textonly)
         CheckItem(myMenus[orbitM], 1, TRUE);
      else
         CheckItem(myMenus[orbitM], 2, TRUE);
      if ( orbitinfo->coordinates == IJK_COORDS )
         CheckItem(myMenus[orbitM], 4, TRUE);
      else if ( orbitinfo->coordinates == PQW_COORDS )
         CheckItem(myMenus[orbitM], 5, TRUE);
      else if ( orbitinfo->coordinates == GEO_COORDS )
         CheckItem(myMenus[orbitM], 6, TRUE);
   }


/* **** Handle disabling menus due to lack of any orbit windows here  */
/* **** Initially enable all menu items that the program supports...  */
   EnableItem( myMenus[fileM], fmNew );
   EnableItem( myMenus[fileM], fmOpen );
   EnableItem( myMenus[fileM], fmClose );
   EnableItem( myMenus[fileM], fmSave );
   EnableItem( myMenus[fileM], fmSaveAs );
   EnableItem( myMenus[fileM], fmRevert );
   EnableItem( myMenus[fileM], fmPrint );

   DisableItem( myMenus[editM], undoCommand );
   DisableItem( myMenus[editM], cutCommand );
   DisableItem( myMenus[editM], copyCommand );
   DisableItem( myMenus[editM], pasteCommand );
   DisableItem( myMenus[editM], clearCommand );


   if ( !MO_AvailWind())      /* There is no room to create              */
   {                          /*          another orbit record           */
      DisableItem ( myMenus[fileM], fmNew );
      DisableItem ( myMenus[fileM], fmOpen );
   }

   if ( !MO_ours( FrontWindow() ))
   {                          /* no open orbit window so must be a DA    */
      DisableItem( myMenus[fileM], fmClose );
      DisableItem( myMenus[fileM], fmSave );
```

```
   DisableItem( myMenus[fileM], fmSaveAs );
   DisableItem( myMenus[fileM], fmRevert );
   DisableItem( myMenus[fileM], fmPrint );
   EnableItem( myMenus[editM], undoCommand );
   EnableItem( myMenus[editM], cutCommand );
   EnableItem( myMenus[editM], copyCommand );
   EnableItem( myMenus[editM], pasteCommand );
   EnableItem( myMenus[editM], clearCommand );
}
else if ( MO_isDirty(FrontWindow())  )
{
   if (MO_isNewFile(FrontWindow()) )
   {                    /* source of the orbit is a file          */
      EnableItem( myMenus[fileM], fmSave );
      EnableItem( myMenus[fileM], fmRevert );
   }
   else                 /* source of orbit is NEW menu item       */
   {
      DisableItem( myMenus[fileM], fmSave );
      DisableItem( myMenus[fileM], fmRevert );
   }
}
else                    /* saved unchanges source file            */
{
   DisableItem( myMenus[fileM], fmRevert );
   DisableItem( myMenus[fileM], fmSave );
}


if ((!MO_ours( FrontWindow())) && (prefer.bar_state == ON))
{                       /* there is no open orbit window*/
   prefer.bar_state = OFF;          /* Turn off the Orbit menu    */
   DisableItem( myMenus[orbitM], 0);
                        /* 0 is the ENTIRE menu disable           */
   DrawMenuBar();/* redraw menu bar                               */
}
else if ((MO_ours( FrontWindow())) && (prefer.bar_state == OFF))
{                       /* there is atleast one open window       */
   prefer.bar_state = ON;           /* Turn on the Orbit menu     */
   EnableItem( myMenus[orbitM], 0);
                        /* 0 is the ENTIRE menu enable            */
   DrawMenuBar();/* redraw menu bar                               */
}

if (prefer.plotting == TRUE)
{                       /* show plot choice via checkmark         */
   DisableItem( myMenus[plotM], plONE );
   DisableItem( myMenus[plotM], plCONT );
   EnableItem( myMenus[plotM], plSTOP );
   CheckItem(myMenus[plotM], prefer.plot_duration, TRUE);
```

```
    }
    else
    {
       if (prefer.time == 0)
       {                        /* No plotting is already in progress     */
          SetItem(myMenus[plotM], plONE,  "\pTimed Plot...");
          SetItem(myMenus[plotM], plCONT, "\pStart Continuous Plot");
       }
       else
       {                        /* This plot is already in progress       */
          SetItem(myMenus[plotM], plONE,  "\pResume Timed Plot...");
          SetItem(myMenus[plotM], plCONT, "\pResume Continuous Plot");
       }
       EnableItem( myMenus[plotM], plONE );
       EnableItem( myMenus[plotM], plCONT );
       DisableItem( myMenus[plotM], plSTOP );
       CheckItem(myMenus[plotM], prefer.plot_duration, FALSE);
    }

}   /* ******************** MO_MaintainMenus ********************** */


    /*******************************************************************
    MO_SetUpCursors : get handle to all cursors used in program
    ******************************************************************* */
    void
MO_SetUpCursors()
{
CursHandle          theCurs;

    theCurs = GetCursor(watchCursor);
    watch = **theCurs;
} /* ******************** MO_SetUpCursors ************************ */


    /*******************************************************************
    MO_Init_Preferences : set up default session values
    ******************************************************************* */
    void
MO_Init_Preferences( prefer )
    Preferences      *prefer;
{

    prefer->changes = TRUE;  /* Initial drawing of menu items          */
    prefer->bar_state = ON;  /* Orbit item in menu bar is on           */
    prefer->showmap = FALSE; /* display the map                        */
    prefer->std_units = 0;   /* metric(0), canonical(1)                */
    prefer->time_comp = 100.0;/* time compression factor (100:1)       */
```

```
    prefer->time = 0L;        /* 0 seconds of wall clock time          */
    prefer->plotting = FALSE;/* not currently computing orbits         */
    prefer->plot_duration = 0;          /* arbitrarily defined         */
    prefer->draw_method = 0; /* plot (0)dots (1)lines (2)both          */
    prefer->first_plot = TRUE;/* first time into current plot seq      */
    prefer->draw_incr = 10.0;/* increment plot every 10 minutes        */
    prefer->showaxes = TRUE; /* show axes when plotting                */
    prefer->obs_lat = 0.0;   /* latitude of observer                   */
    prefer->obs_lon = 0.0;   /* longitude of observer                  */

}   /* ******************** MO_Init_Preferences ********************** */


    /*****************************************************************
    MO_Init_Globes : load globe pictures into memory
    ***************************************************************** */
    void
MO_Init_Globes()
{
PicHandle           globeHand;
Rect                tempRect;
int                 bmsize, i;
GrafPtr             tempPort, savePort;
    MO_Wait();              /* tell user it will take a while          */

    GetPort(&savePort);     /* save existing grafport                  */
    tempPort = (GrafPtr) NewPtr(sizeof(GrafPort) );
    OpenPort(tempPort);     /* does set for special grafport           */

    for (i=0; i<MAXGLOBES ; i=i+1)
    {
       globeHand = GetPicture(GlobeID+i);
                            /* Get each picture item                   */
       tempRect = (**globeHand).picFrame;
       ClipRect(&tempRect);  /* avoid bug from Apple TN #59            */
       PortSize(tempRect.right-tempRect.left,
                tempRect.bottom-tempRect.top);

       bmsize = 1 + (tempRect.right-tempRect.left) / 8;
       if ((bmsize % 2) != 0)
          bmsize = bmsize+1; /* an even number of bytes                */
       globePics[i].rowBytes = bmsize;
       bmsize = bmsize * (tempRect.bottom-tempRect.top);
       globePics[i].baseAddr = NewPtr(bmsize);
       globePics[i].bounds = tempRect;
       SetPortBits( &(globePics[i]) );

       EraseRect(&tempRect); /* clear the new bitmap and draw           */
       DrawPicture(globeHand, &tempRect );
```

```
    }

    SetPort(savePort);          /* restore old grafport                    */
    ClosePort(tempPort);        /* don't leave the port around...          */

    InitCursor();               /* reset the cursor                        */

}   /* ********************** MO_Init_Globes ************************** */
```

```
/*********************************************************************

          FILENAME            :  MacOrbits.fm.c
          DESCRIPTION         :  file manager interface for MacOrbits.c
          ENVIRONMENT         :  Macintosh SE 1Mb
                                 LightSpeed™ C v2.15
          AUTHOR              :  Captain Kenneth L. BEUTEL USMC
                                 (some portions copyright THINK TECHNOLOGIES™)
          ADVISORS            :  Prof. Dan Davis
                                 Prof. Dan Boger
                                 Naval Postgraduate School, Monterey CA
          REMARKS             :  Contains Mac specific file system
                                 instructions
          VERSION             :  0.9 (3/6/88)

          CHANGES             :  3/6/88 Formatted for MacWrite conversion
*********************************************************************** */

#include "QuickDraw.h"
#include "MacTypes.h"
#include "FontMgr.h"
#include "WindowMgr.h"
#include "MenuMgr.h"
#include "TextEdit.h"
#include "DialogMgr.h"
#include "EventMgr.h"
#include "DeskMgr.h"
#include "FileMgr.h"
#include "ToolboxUtil.h"
#include "ControlMgr.h"
#include "StdFilePkg.h"
#include "stdio.h"

#include "LightSpeed Disk:Thesis C f:StdLib.h"

#include "MacOrbits.h"
#include "MacOrbits.proto.h"

extern  Preferences prefer; /* preferences for how things look        */
extern  MenuHandle  myMenus[specialM + 1];


   /*********************************************************************
   MO_File : handles all File menu generated events
   ********************************************************************* */
   int
MO_File( item, theFileName, theVRefNum, myWindow)
   int               item;
   char              *theFileName;
   int               *theVRefNum;
```

```
   WindowPtr         myWindow;
{
int                  vRef, refNum, resultCode;
Str255               fn;     /* text file name                          */
Str255               windName;/* name of topmost window                 */
Str255               appname;/* application to transfer to              */
WindowPtr            oldWindow;/* saves last window pointed to           */
int                  slotnum;

   switch (item)
   {
   case fmNew:
      oldWindow = myWindow;
      MO_CreateWindow(&myWindow, &slotnum);
      if (myWindow != NIL)
      {
         MO_fmNew( &myWindow, slotnum );
         MO_SetDirty(myWindow, TRUE);
      }
      else
      {
         MO_Generic("\pSorry, max windows already open.","\p");
         myWindow = oldWindow;
      }
   break;
   case fmOpen:
      if (MO_OldFile( &fn, &vRef ))
         if (!MO_DuplicWind(fn))
            if (FSOpen( &fn, vRef, &refNum)==noErr)
            {
               MO_CreateWindow(&myWindow, &slotnum);
               if (myWindow != NIL)
               {
                  if (MO_ReadFile( refNum, myWindow)==noErr)
                  {
                     *theVRefNum = vRef;
                     MO_pStrCopy(fn, theFileName);
                     SetWTitle(myWindow, theFileName);

                     SetItem(myMenus[windowM], slotnum + 5, theFileName);
                     EnableItem(myMenus[windowM], slotnum + 5);
                     ShowWindow( myWindow );
                  }
               }
               else
               {
                  MO_Generic("\pSorry, max windows already open.",
                             "\p","\p","\p");
                  myWindow = oldWindow;
               }
```

```
            FSClose(refNum);
         }
         else
            MO_Generic( "\pError opening ", fn );
break;
case fmClose:
   if (MO_isDirty(myWindow))
   {
      GetWTitle(myWindow, &windName);
      switch (MO_Advise(&windName))
      {
      case dlgSave:
         if (MO_isNewFile(myWindow) )
         {                    /* No filename so it is a new window      */
            GetWTitle(myWindow, &fn);
            if (!MO_SaveAs(&fn, &vRef, myWindow))
               return(FALSE);
         }
         else if (!MO_SaveFile( theFileName, *theVRefNum, myWindow ))
            return(FALSE);
         MO_RemoveWindow(&myWindow);
                              /* Only remove window for saved file      */
      break;
      case dlgCancel:
         return(FALSE);
      case dlgDiscard:
         MO_RemoveWindow(&myWindow);
      break;
      }
   }
   else
   {
      MO_RemoveWindow(&myWindow);
   }
break;
case fmSave:
   if (!MO_isNewFile(myWindow))
      MO_SaveFile(theFileName, *theVRefNum, myWindow);
   else                       /* Not from an old file                   */
   {
      GetWTitle(myWindow, &fn);
      if (MO_SaveAs( &fn, &vRef, myWindow ))
         *theVRefNum = vRef;
   }
break;
case fmSaveAs:
   GetWTitle(myWindow, &fn);
   if (MO_SaveAs( &fn, &vRef, myWindow ))
      *theVRefNum = vRef;
break;
```

```
    case fmRevert:
       if ((!MO_isNewFile(myWindow) )
           && (FSOpen( &theFileName, *theVRefNum, &refNum) == noErr))
       {
          MO_SetDirty(myWindow, !MO_ReadFile( refNum, myWindow) );
          FSClose(refNum);
       }
       ShowWindow( myWindow );
       MO_UpdateWindow( myWindow );
    break;
    case fmPageSetUp:
       MO_PageSetUp();
    break;
    case fmPrint:
       MO_PrintOrbitText(myWindow);
    break;
    case fmXFER:              /* get names of all open windows          */
       while (MO_FirstWindow() != NIL)
       {                      /* recursive call to close files          */
          SelectWindow(MO_FirstWindow());
          MO_File(fmClose, theFileName, &*theVRefNum, MO_FirstWindow() );
       }
       if (MO_FirstWindow() == NIL)
          if ( MO_OldApps( &appname, &vRef) )
          {
             SetVol("\p", vRef);
                               /* make appl. volume the curr  one        */
             Launch(0L, appname);
                               /* and launch the app                     */
           }
    break;
    case fmQuit:              /* get  filenames of all open windows     */
       while (MO_FirstWindow() != NIL)
       {                      /* recursive call to close files          */
          SelectWindow(MO_FirstWindow());
          MO_File(fmClose, theFileName, &*theVRefNum, MO_FirstWindow() );
       }
       if (MO_FirstWindow() == NIL)
          ExitToShell();
    break;
    }

    return(TRUE);
}   /* ********************* MO_File ******************************** */


    /*************************************************************
    MO_SaveAs : handles save as for topmost window
    ************************************************************* */
```

```
   int
MO_SaveAs( name, vRefnum, myWindow )
   Str255            *name;
   int               *vRefnum;
   WindowPtr         myWindow;
{
OrbitInfo            *orbitinfo;
int                  refNum;

   orbitinfo = (OrbitInfo *) (( (WindowPeek) (myWindow))->refCon);
   if (MO_NewFile(&*name, &*vRefnum))
      if (MO_CreateFile(&*name, &*vRefnum, &refNum))
      {
         MO_pStrCopy(*name, orbitinfo->orbitdata.name);
         SetWTitle(myWindow, orbitinfo->orbitdata.name);
         SetItem(myMenus[windowM],
                   (orbitinfo->slotnum) + 5, orbitinfo->orbitdata.name);

         MO_WriteFile( refNum, myWindow );
         FSClose( refNum );
         MO_SetDirty(myWindow, FALSE);
         return(TRUE);
      }
      else
      {
         MO_Generic("\pError creating file ", name);
         return(FALSE);
      }

}   /* ********************* MO_SaveAs ****************************** */



   /*********************************************************************
   MO_SaveFile : handles save of topmost window
   ********************************************************************* */
   int
MO_SaveFile( name, vRefNum, myWindow )
   char              *name;
   int               vRefNum;
   WindowPtr         myWindow;
{
int refNum;

   if (FSOpen( &*name, vRefNum, &refNum )==noErr)
   {
      MO_WriteFile( refNum, myWindow );
      MO_SetDirty(myWindow, FALSE);
      FSClose( refNum );
      return(TRUE);
```

```
    }
    else
    {
       MO_Generic("\pError opening file ", name);
       return(FALSE);
    }
}    /* ******************** MO_SaveFile ***************************** */


    /*******************************************************************
    MO_Advise : std Save,Discard,Cancel choices in a dialog box
    ******************************************************************* */
    int
MO_Advise( filename )
    Str255 *filename;
{
    ParamText(*filename, "\p", "\p", "\p");
    return( CautionAlert(DirtyFileID, 0L) );
} /* ******************* MO_Advise ******************************** */


    /*******************************************************************
    MO_NewFile : gets name and volume information for new document
    ******************************************************************* */
    int
MO_NewFile( name, vRefnum )
    Str255            *name;
    int               *vRefnum;
{
static  Point        SFPwhere = { 106, 104 };
static  SFReply      MO_reply;

    SFPutFile(SFPwhere, "\p", name, 0L, &MO_reply);
    if (MO_reply.good)
    {
       MO_pStrCopy(MO_reply.fName, &*name);
       *vRefnum = MO_reply.vRefNum;
       return(TRUE);
    }
    else
       return(FALSE);
}    /* ******************** MO_NewFile ***************************** */


    /*******************************************************************
    MO_OldFile : gets name and volume information for existing document
    ******************************************************************* */
```

```
   int
MO_OldFile( name, vRefnum )
   Str255            *name;
   int               *vRefnum;
{
SFTypeList           myTypes;
static  SFReply      MO_reply;
static  Point        SFGwhere = { 90, 82 };

   myTypes[0]='ORBT';
   SFGetFile( SFGwhere, "\p", 0L, 1, myTypes, 0L, &MO_reply );
   if (MO_reply.good)
   {
      MO_pStrCopy( MO_reply.fName, name );
      *vRefnum = MO_reply.vRefNum;
      return(TRUE);
   }
   else
      return(FALSE);
}   /* ******************** MO_OldFile ***************************** */



   /*********************************************************************
   MO_OldApps : gets name and volume information for existing application
   ********************************************************************* */
   int
MO_OldApps( name, vRefnum )
   Str255            *name;
   int               *vRefnum;
{
SFTypeList           myTypes;
static  SFReply      MO_reply;
static  Point        SFGwhere = { 90, 82 };

   myTypes[0]='APPL';
   SFGetFile( SFGwhere, "\p", 0L, 1, myTypes, 0L, &MO_reply );
   if (MO_reply.good)
   {
      MO_pStrCopy( MO_reply.fName, name );
      *vRefnum = MO_reply.vRefNum;
      return(TRUE);
   }
   else
      return(FALSE);
}   /* ******************** MO_OldApps ***************************** */



   /*********************************************************************
```

```
   MO_CreateFile : handles creation of a new file
   ********************************************************************** */
   int
MO_CreateFile( name, vRefnum, theRef )
   Str255            *name;
   int               *vRefnum;
   int               *theRef;
{
int resultCode;             /* i/o result code                          */

   resultCode=Create(name, *vRefnum, '????', 'ORBT');
   if ((resultCode==noErr) || (resultCode==dupFNErr))
      resultCode = FSOpen( name, *vRefnum, theRef );

   return( (resultCode==noErr) || (resultCode=dupFNErr) );
}   /* ********************** MO_CreateFile ************************** */



   /*********************************************************************
   MO_WriteFile : handles writing data into a file
   ********************************************************************** */
   int
MO_WriteFile( refNum, myWindow )
   int               refNum;
   WindowPtr         myWindow;
{
long                 num;
char                 buffer[80], temp[80];
OrbitInfo            *orbitinfo;
int                  resultCode;/* input output return code             */

   orbitinfo = (OrbitInfo *) (( (WindowPeek) (myWindow))->refCon);

   MO_pStrCopy(gl_gorbn(orbitinfo->orbitdata), temp);
   PtoCstr(&temp);
   sprintf(buffer, "%s\n", temp );
   num = strlen(buffer);
   resultCode = FSWrite( refNum, &num, buffer );

   sprintf(buffer, "Semi-major: %G\n", gl_gorba(orbitinfo->orbitdata) );
   num = strlen(buffer);
   resultCode = FSWrite( refNum, &num, buffer );

   sprintf(buffer, "Eccentricity: %G\n", gl_gorbe(orbitinfo->orbitdata) );
   num = strlen(buffer);
   resultCode = FSWrite( refNum, &num, buffer );

   sprintf(buffer, "Inclination: %G\n",
          cv_gangd(gl_gorbi(orbitinfo->orbitdata)) );
```

```
   num = strlen(buffer);
   resultCode = FSWrite( refNum, &num, buffer );

   sprintf(buffer, "Mean Anomaly: %G\n",
          cv_gangd(gl_gorbm(orbitinfo->orbitdata)) );
   num = strlen(buffer);
   resultCode = FSWrite( refNum, &num, buffer );

   sprintf(buffer, "Arg of Peri: %G\n",
          cv_gangd(gl_gorbp(orbitinfo->orbitdata)) );
   num = strlen(buffer);
   resultCode = FSWrite( refNum, &num, buffer );

   sprintf(buffer, "Long of ASN: %G\n",
          cv_gangd(gl_gorbl(orbitinfo->orbitdata)) );
   num = strlen(buffer);
   resultCode = FSWrite( refNum, &num, buffer );

   sprintf(buffer, "Epoch: %G\n", gl_gorbt(orbitinfo->orbitdata) );
   num = strlen(buffer);
   resultCode = FSWrite( refNum, &num, buffer );

   sprintf(buffer, "Julian Date: %G\n", (orbitinfo->orbitdata).date );
   num = strlen(buffer);
   resultCode = FSWrite( refNum, &num, buffer );
}   /* ********************* MO_WriteFile *************************** */



   /*********************************************************************
   MO_ReadFile : handles reading data from a file
   ********************************************************************* */
   int
MO_ReadFile( refNum, myWindow)
   int               refNum;
   WindowPtr         myWindow;
{
char                 buffer[80];
char                 orbname[80];
int                  resultCode;
Real                 e, date;
Dist                 a;
Angle                i, mean, peri, long_asn;
Time                 time;
char                 temp[30];
OrbitInfo            *orbitinfo;

   resultCode = MO_ReadLine( buffer, refNum);
   if (resultCode==noErr)   /* get orbit name                          */
      sscanf(buffer, "%s", orbname);
```

```
    resultCode = MO_ReadLine( buffer, refNum);
    if (resultCode==noErr)   /* get orbit semimajor axis                */
       sscanf(buffer, "%*s %lG", &a);

    resultCode = MO_ReadLine( buffer, refNum);
    if (resultCode==noErr)   /* get orbit eccentricity                  */
       sscanf(buffer, "%*s %lG", &e);

    resultCode = MO_ReadLine( buffer, refNum);
    if (resultCode==noErr)   /* get orbit inclination                   */
       sscanf(buffer, "%*s %lG", &i);

    resultCode = MO_ReadLine( buffer, refNum);
    if (resultCode==noErr)   /* get orbit mean anomaly                  */
       sscanf(buffer, "%*s %*s %lG", &mean);

    resultCode = MO_ReadLine( buffer, refNum);
    if (resultCode==noErr)   /* get orbit argument of perigee           */
       sscanf(buffer, "%*s %*s %*s %lG", &peri);

    resultCode = MO_ReadLine( buffer, refNum);
    if (resultCode==noErr)   /* get orbit longitude of ASN              */
       sscanf(buffer, "%*s %*s %*s %lG", &long_asn);

    resultCode = MO_ReadLine( buffer, refNum);
    if (resultCode==noErr)   /* get orbit epoch time                    */
       sscanf(buffer, "%*s %lG", &time);

    resultCode = MO_ReadLine( buffer, refNum);
    if (resultCode==noErr)   /* get orbit epoch date                    */
       sscanf(buffer, "%*s %*s %lG", &date);

    if (resultCode==noErr)
    {
       orbitinfo = (OrbitInfo *) (( (WindowPeek) (myWindow))->refCon);
       gl_sorbt(orbname, a, e, cv_sangd(i), cv_sangd(mean), cv_sangd(peri),
                cv_sangd(long_asn), time, date, &(orbitinfo->orbitdata) );
       orbitinfo->newfile = FALSE;
                           /* file is an old file from disk           */
    }

    return( resultCode==eofErr );
}   /* ********************* MO_ReadFile ***************************** */



    /*****************************************************************
    MO_ReadLine : handles reading a line of data from a file
    ***************************************************************** */
```

```
   int
MO_ReadLine(buffer, refNum)
   char              *buffer;
   int               refNum;
{
long                 buffer_pos;
char                 singlechar;
long                 count;
char                 errormsg[30];
int                  resultCode;

   buffer_pos = 0;
   do
   {
      count = 1;
      resultCode = FSRead( refNum, &count, &singlechar );
      if (resultCode != noErr)
      {
         sprintf(errormsg, "Sorry, an ioError = &d", resultCode);
         CtoPstr(&errormsg);
         MO_Generic(errormsg,"\pOccurred");
         return(resultCode);
      }
      buffer[buffer_pos] = singlechar;
      buffer_pos = buffer_pos + 1;
   } while ((buffer_pos < 80) && (buffer[buffer_pos-1] != '\n'));
   buffer[buffer_pos-1] = '\0';
                        /* delimit end of string                    */
   return(resultCode);

}   /* ********************* MO_ReadLine ***************************** */



   /*********************************************************************
   MO_fmNew : Via dialog get the info to setup a new orbit
   ********************************************************************* */
   void
MO_fmNew( myWindow, slotnum)
   WindowPtr         *myWindow;
   int               slotnum;
{
DialogPtr myDialog;
Boolean              not_complete;
int                  itemhit;
int                  dummy;
Rect                 rect, button_rect;
Handle               OK_button, item_a, item_e, item_name, item_i, item_m;
Handle               item_peri, item_long, item_time, item_date, item_units;
Str255               str;
```

```
OrbitInfo           *orbitinfo;
Real                a, e, i, mean, peri, long_asn, time, date;
Str255              orbname;

   orbitinfo = (OrbitInfo *) (( (WindowPeek) (*myWindow))->refCon);
   myDialog = GetNewDialog(NewOrbitID, NIL, INFRONT);

   GetDItem(myDialog,  1, &dummy, &OK_button, &button_rect);
   GetDItem(myDialog,  3, &dummy, &item_name, &rect);
   GetDItem(myDialog,  4, &dummy, &item_a,    &rect);
   GetDItem(myDialog,  5, &dummy, &item_e,    &rect);
   GetDItem(myDialog, 10, &dummy, &item_i,    &rect);
   GetDItem(myDialog, 27, &dummy, &item_m,    &rect);
   GetDItem(myDialog, 12, &dummy, &item_peri, &rect);
   GetDItem(myDialog, 14, &dummy, &item_long, &rect);
   GetDItem(myDialog, 16, &dummy, &item_time, &rect);
   GetDItem(myDialog, 24, &dummy, &item_date, &rect);
   if (prefer.std_units == 1)          /* Use canonical units        */
   {                                   /* (default is Metric Units)  */
      GetDItem(myDialog, 22, &dummy, &item_units, &rect);
      SetIText(item_units, "\p(hergs)");
      GetDItem(myDialog, 17, &dummy, &item_units, &rect);
      SetIText(item_units, "\p(Earth radii)");
   }
   SelIText(myDialog, 3, 0, 999);
   not_complete = TRUE;
   itemhit = 0;
   MO_OutlineButton( button_rect, myDialog );

   while ((not_complete==TRUE) && (itemhit!=2))
   {
      do
      {
         ModalDialog(NIL, &itemhit);
      } while ((itemhit != 1) && (itemhit != 2));
                                   /* OK or Cancel Button        */
      if (itemhit == 1)            /* OK Button ->check results  */
      {
         GetIText(item_name, &(orbname) );
         GetIText(item_a, &str);
            a = MO_pStr2Num(&str);
         GetIText(item_e, &str);
            e = MO_pStr2Num(&str);
         GetIText(item_i, &str);
            i = MO_pStr2Num(&str);
         GetIText(item_m, &str);
           mean = MO_pStr2Num(&str);
         GetIText(item_peri, &str);
            peri = MO_pStr2Num(&str);
         GetIText(item_long, &str);
```

```
            long_asn = MO_pStr2Num(&str);
         GetIText(item_time, &str);
            time = MO_pStr2Num(&str);
         GetIText(item_date, &str);
            date = MO_pStr2Julian(&str);

                              /* check if the values are valid          */
         if (orbname[0] != 0)
            if (a >= 0)
               if (e >= 0)
                  if (i >= 0)
                  {
                     not_complete = FALSE;
                     if (prefer.std_units == 1)
                     {      /* convert from canonical units           */
                        a = a * 6378.6;
                     }
                     gl_sorbt(orbname, a, e, cv_sangd(i), cv_sangd(mean),
                              cv_sangd(peri), cv_sangd(long_asn), time,
                              date, &(orbitinfo->orbitdata) );
                  }
         if (not_complete)
            SysBeep(3);

         if (MO_DuplicWind(orbname))
            not_complete = TRUE;

      }                       /* end if OK Button                       */
   }                          /* endwhile not complete                  */

   DisposDialog(myDialog);

   if (not_complete == FALSE)/* it is complete so add the window       */
   {
      SetWTitle( *myWindow, orbitinfo->orbitdata.name);
      ShowWindow( *myWindow );

      SetItem(myMenus[windowM], slotnum + 5, orbitinfo->orbitdata.name);
      EnableItem(myMenus[windowM], slotnum + 5);
   }
   else                       /* not complete so remove the window      */
   {
      MO_RemoveWindow( &*myWindow );
   }
}   /* ********************** MO_fmNew ***************************** */
```

```
/*********************************************************************

          FILENAME            :  MacOrbits.map.c
          DESCRIPTION         :  handles setup & drawing of cartesian map
          ENVIRONMENT         :  Macintosh SE 1Mb
                                 LightSpeed™ C v2.15
          AUTHOR              :  Captain Kenneth L. BEUTEL USMC
          ADVISORS            :  Prof. Dan Davis
                                 Prof. Dan Boger
                                 Naval Postgraduate School, Monterey CA
          REMARKS             :  Contains Mac specific drawing commands
          VERSION             :  0.9 (3/6/88)

          CHANGES             :  3/6/88 Formatted for MacWrite conversion
********************************************************************* */

#include "QuickDraw.h"
#include "MacTypes.h"
#include "FontMgr.h"
#include "WindowMgr.h"
#include "MenuMgr.h"
#include "TextEdit.h"
#include "DialogMgr.h"
#include "EventMgr.h"
#include "DeskMgr.h"
#include "FileMgr.h"
#include "ToolboxUtil.h"
#include "ControlMgr.h"
#include "stdio.h"

#include "LightSpeed Disk:Thesis C f:StdLib.h"

#include "MacOrbits.h"
#include "MacOrbits.proto.h"

extern Preferences prefer;  /* preferences for how things look        */


   /*********************************************************************
   MO_CreateMap : get bitmap of picture for "stamping" on screen later
   ********************************************************************* */
   void
MO_CreateMap(bm, mapWindow)
   BitMap            *bm;
   WindowPtr         *mapWindow;
{
GrafPtr              mapPort;
GrafPtr              savePort;
PicHandle            mapHandle;
Rect                 tempRect;
```

```
int                 bmsize; /* bit map size                                */

   if (*mapWindow != NIL)
      return;               /* done - A map window already exists        */

   MO_Wait();               /* tell user it will take a while            */
   GetPort(&savePort);
   mapHandle = GetPicture(MapID);
                            /* First get the picture drawn               */
   mapPort = (GrafPtr) NewPtr(sizeof(GrafPort) );
   OpenPort(mapPort);       /* does set for special grafport             */


   tempRect = (**mapHandle).picFrame;
   ClipRect(&tempRect);     /* avoid bug from Apple TN #59               */
   PortSize(tempRect.right-tempRect.left, tempRect.bottom-tempRect.top);

   bmsize = 1 + (tempRect.right-tempRect.left) / 8;
   if ((bmsize % 2) != 0)
      bmsize = bmsize+1;    /* an even number of bytes                   */
   (*bm).rowBytes = bmsize;
   bmsize = bmsize * (tempRect.bottom-tempRect.top);
   (*bm).baseAddr = NewPtr(bmsize);
   (*bm).bounds = tempRect;
   SetPortBits(&*bm);

   EraseRect(&tempRect);    /* clear the new bitmap and draw             */
   DrawPicture(mapHandle, &tempRect );

   SetPort(savePort);       /* restore old grafport                      */
   ClosePort(mapPort);      /* don't leave the port around...            */

   *mapWindow = GetNewWindow(MapID, NIL, NIL);

   InitCursor();            /* reset the cursor                          */

}   /* ********************** MO_CreateMap ************************** */



   /*****************************************************************
   MO_DeleteMap : deallocate bitmap and window to free memory
   ***************************************************************** */
   void
MO_DeleteMap(bm, mapWindow)
   BitMap           *bm;
   WindowPtr        *mapWindow;
{
GrafPtr             savePort;
Rect                tempRect;
```

```
int                 bmsize; /* bit map size                                */

   if (*mapWindow == NIL)
      return;               /* No map window  exists                       */

   DisposPtr( (*bm).baseAddr );
   (*bm).baseAddr = NIL;
   (*bm).rowBytes = 0;

   DisposeWindow( *mapWindow );
   *mapWindow = NIL;
}   /* ********************** MO_DeleteMap ************************** */



   /*********************************************************************
   MO_DrawMap : stamp the map into the current window inside dstRect
   ********************************************************************* */
   void
MO_DrawMap(bm, mapWindow)
   BitMap           bm;
   WindowPtr        mapWindow;
{
static Rect         dstRect =
                      { MAP_TOP+9, MAP_LEFT, MAP_TOP+9+160, MAP_LEFT+416 };
GrafPtr             savePort;
Rect                r;      /* entire map rect incl. poles                 */

   if (prefer.showmap != TRUE)
      return;

   GetPort( &savePort );
   SetPort( mapWindow );
   InvalRect( &(mapWindow->portRect) );
   BeginUpdate( mapWindow );

   MoveTo(120, 30);         /* horiz, vert                                 */
   TextFont(systemFont);
   DrawString("\p World Geographic Coordinate System");

   CopyBits(&bm,&(mapWindow->portBits),&((bm).bounds),&dstRect,srcCopy, NIL);

   r.top    = MO_LattoPixel(90.0);
                            /* 90°N.                                       */
   r.left   = MO_LontoPixel(181.0)-3;
                            /* 181°E.                                      */
   r.bottom = MO_LattoPixel(-90.0);
                            /* 90°S.                                       */
   r.right = MO_LontoPixel(180.0)+4;
                            /* 180°E.                                      */
```

```
   FrameRect(&r);              /* Frame the map                              */


                               /* Draw Monterey as a Square Box              */
   r.left = MO_LontoPixel(360.0-120.0) -2;
   r.right = r.left + 4;
   r.top  = MO_LattoPixel(36.0)-2;
   r.bottom= r.top + 4;
   FillRect(&r, black);

   EndUpdate( mapWindow );
   SetPort( savePort );
}   /* ********************** MO_DrawMap *************************** */



   /*****************************************************************
   MO_DrawMapUpdate : add satellite pos to global map
   ***************************************************************** */
   void
MO_DrawMapUpdate( orbitinfo, mapWindow )
   OrbitInfo         orbitinfo;
   WindowPtr         mapWindow;
{
GrafPtr              savePort;
Rect                 r;

   GetPort( &savePort );
   SetPort( mapWindow );
   switch (prefer.draw_method)
   {
   case 0:                     /* dots only                                  */
      r.top = MO_LattoPixel( cv_gangd(orbitinfo.geo.latitude) );
      r.bottom = r.top + 5;
      r.left = MO_LontoPixel( cv_gangd(orbitinfo.geo.longitude));
      r.right = r.left + 5;
      FillOval(&r, black);
   break;
   case 1:                     /* lines only                                 */
      if (prefer.first_plot != TRUE)
      {
         MoveTo(MO_LontoPixel( cv_gangd(orbitinfo.last_geo.longitude)),
                MO_LattoPixel( cv_gangd(orbitinfo.last_geo.latitude) ));
         r.top = MO_LattoPixel( cv_gangd(orbitinfo.geo.latitude) );
         r.left = MO_LontoPixel( cv_gangd(orbitinfo.geo.longitude));
         if ((orbitinfo.last_geo.longitude <= PI)
             && (orbitinfo.geo.longitude >= PI))
            break;           /* early break due to changing map edge       */
         LineTo(r.left, r.top);
      }
```

```
   break;
   case 2:                    /* both lines and dots                    */
      if (prefer.first_plot != TRUE)
      {
         MoveTo(MO_LontoPixel( cv_gangd(orbitinfo.last_geo.longitude)),
                MO_LattoPixel( cv_gangd(orbitinfo.last_geo.latitude) ));
         r.top = MO_LattoPixel( cv_gangd(orbitinfo.geo.latitude) );
         r.left = MO_LontoPixel( cv_gangd(orbitinfo.geo.longitude));

         if ((orbitinfo.last_geo.longitude <= PI)
             && (orbitinfo.geo.longitude >= PI))
            break;           /* early break due to changing map edge   */
         LineTo(r.left, r.top);
         r.top = r.top - 2; /* center the dot                          */
         r.bottom = r.top + 5;
         r.left = r.left - 2;/* center the dot                         */
         r.right = r.left + 5;
         FillOval(&r, black);
      }
      else                     /* starting so just draw a dot           */
      {
         r.top = MO_LattoPixel( cv_gangd(orbitinfo.geo.latitude) ) - 2;
         r.bottom = r.top + 5;
         r.left = MO_LontoPixel( cv_gangd(orbitinfo.geo.longitude)) - 2;
         r.right = r.left + 5;
         FillOval(&r, black);
      }
   break;
   }
   SetPort( savePort );

}   /* ******************** MO_DrawMapUpdate ************************ */


   /*****************************************************************
   MO_LattoPixel : compute the pixel nearest to North(>0) or South(<0)
   ***************************************************************** */
   int
MO_LattoPixel(lat)
   double            lat;
{
int                  temp;

   temp = ((90-lat) * PIX_LAT) + MAP_TOP;
   return( temp );

}   /* ********************** MO_LattoPixel ************************** */
```

```
   /*****************************************************************
   MO_LontoPixel : compute the pixel nearest to East Longitude
                   where west (left) map edge is 181° E.Longitude and
                   east (right) map edge is 180° E. Long.
   ***************************************************************** */
   int
MO_LontoPixel(lon)
   double            lon;
{
int                  temp;

   if ((0.0 <= lon) && (lon <= 180.0))
      temp = (lon * PIX_LON) + (MAP_LEFT + 416/2);
   else if ((180.0 < lon) && (lon <= 360.0))
      temp = ((lon-180) * PIX_LON) + MAP_LEFT ;
   else
      temp = 0;
   return( temp );

}   /* ********************** MO_LontoPixel ************************** */
```

```
/*********************************************************************

          FILENAME            :  MacOrbits.menu.c
          DESCRIPTION         :  menu manager for other than first 3 menus
          ENVIRONMENT         :  Macintosh SE 1Mb
                                 LightSpeed™ C v2.15
          AUTHOR              :  Captain Kenneth L. BEUTEL USMC
          ADVISORS            :  Prof. Dan Davis
                                 Prof. Dan Boger
                                 Naval Postgraduate School, Monterey CA
          REMARKS             :  plot menu code found in MacOrbits.pl.c
          VERSION             :  0.9 (3/6/88)

          CHANGES             :  3/6/88 Formatted for MacWrite conversion
*********************************************************************** */

 include "QuickDraw.h"
#include "MacTypes.h"
#include "FontMgr.h"
#include "WindowMgr.h"
#include "MenuMgr.h"
#include "TextEdit.h"
#include "DialogMgr.h"
#include "EventMgr.h"
#include "DeskMgr.h"
#include "FileMgr.h"
#include "ToolboxUtil.h"
#include "ControlMgr.h"
#include "StdFilePkg.h"
#include "stdio.h"

#include "LightSpeed Disk:Thesis C f:StdLib.h"

#include "MacOrbits.h"
#include "MacOrbits.proto.h"

extern Preferences  prefer; /* preferences for how things look        */
extern WindowPtr    mapWindow;/* the window that draws world map      */
extern BitMap       map_bm; /* where the original map is drawn        */



   /*********************************************************************
   MO_OrbitsMenu : info on how to display orbit
   ********************************************************************* */
   void
MO_OrbitsMenu(theItem, theWind)
   int              theItem;
   WindowPtr        theWind;
{
```

```
Str255              str;
OrbitInfo           *orbitinfo;
Rect                plotRect;

   orbitinfo = (OrbitInfo *) (( (WindowPeek) (theWind))->refCon);
   switch (theItem)
   {
   case 1:                  /* **** DISPLAY AS TEXT ****              */
      orbitinfo->textonly = TRUE;
   break;
   case 2:                  /* **** DISPLAY GRAPHICALLY ****/
      orbitinfo->textonly = FALSE;
                            /* fix to erase textbox of graphic wind   */
      plotRect = theWind->portRect;
      plotRect.bottom = plotRect.top + (2*BOX_V+4);
      EraseRect(&plotRect);
   break;
   case 4:                  /* ****IJK COORDS ****                    */
      orbitinfo->coordinates = IJK_COORDS;
   break;
   case 5:                  /* **** PQW COORDS ****                   */
      orbitinfo->coordinates = PQW_COORDS;
   break;
   case 6:                  /* **** GEOGRAPHIC COORDS ****            */
      orbitinfo->coordinates = GEO_COORDS;
   break;
   default:
      NumToString(theItem, str);
      MO_Generic( "\pUnimplemented Orbit Menu item:", str );
   }
   prefer.changes = TRUE;   /* Adjust menu bar approriately           */
   InvalRect(&(theWind->portRect));

}   /* ********************* MO_OrbitsMenu ************************** */



   /*****************************************************************
   MO_PlotMenu : info on how to display orbit
   ***************************************************************** */
   void
MO_PlotMenu(theItem)
   int              theItem;
{
Str255              str;

   switch (theItem)
   {
   case plONE:              /* **** PLOT FOR SPECIFIED TIME ****      */
      MO_SetPlotDuration();
```

```
      break;
      case plCONT:              /* **** PLOT CONTINUOUSLY ****            */
         prefer.plot_duration = plCONT;
         prefer.plotting = TRUE;
         prefer.first_plot = TRUE;
                                /* initiating new plot sequence           */
         prefer.elapsed_time = TickCount();
         prefer.changes = TRUE;/* Update the menu items appropriately    */
      break;
      case plRESET:             /* ****RESET PLOT ****                    */
         prefer.time = 0;       /* simply reset the counters...           */
         prefer.elapsed_time = TickCount();
         MO_ForceUpdate();      /* and force redraw of whole screen       */
         MO_DrawMap(map_bm, mapWindow);
      break;
      case plSTOP:              /* ****STOP PLOTTING ****                 */
         prefer.plotting = FALSE;
         prefer.changes = TRUE;/* Update the menu items appropriately    */
      break;
      default:
         NumToString(theItem, str);
         MO_Generic( "\pUnimplemented Plot Menu item:", str );
      }
   }   /* ********************* MO_PlotMenu ***************************** */



      /*********************************************************************
      MO_SetPlotDuration : select amount of time to plot for
      ********************************************************************* */
      void
   MO_SetPlotDuration()
   {
   DialogPtr myDialog;
   int                  itemhit;
   int                  dummy;
   Handle               time_text, ok_button;
   double               time_val;
   Rect                 rect;
   Str255               str;

      myDialog = GetNewDialog(PlotDurationID, NIL, INFRONT);
      GetDItem(myDialog, 4, &dummy, &time_text, &rect);
                           /* wall clock plot time (minutes)             */
      GetDItem(myDialog, 1, &dummy, &ok_button, &rect);
      MO_OutlineButton( rect, myDialog );
      SelIText(myDialog, 4, 0, 99);
                           /* Hilight text                               */
      itemhit = 0;
```

```
   while ((itemhit != 1) && (itemhit != 2))
   {
      ModalDialog(NIL, &itemhit);
      if (itemhit == 1)
      {
         GetIText(time_text, &str);
         time_val = MO_pStr2Num(&str);
         if (time_val <= 0.0)
            itemhit = 0;     /* invalidate OK button selection          */
      }
   }

   if (itemhit == 1)         /* OK button so set changes                */
   {
      prefer.plot_duration = plONE;
      prefer.plotting = TRUE;
      prefer.first_plot = TRUE;
                             /* initiating new plot sequence            */
      prefer.elapsed_time = TickCount();
      prefer.changes = TRUE;/* Update the menu items appropriately      */
      prefer.stop_time = time_val * 60.0;
                             /* convert minutes to secs                 */
   }
   DisposDialog(myDialog);
}   /* ********************* MO_SetPlotDuration ********************** */



   /*************************************************************************
   MO_WindowMenu : info on how to configure world
   ************************************************************************* */
   void
MO_WindowMenu(theItem)
   int               theItem;
{
Str255               str;
WindowPtr            current;

   switch (theItem)
   {
   case 1:                   /* **** SHOW/HIDE GLOBAL MAP ****          */
      if (prefer.showmap)
      {
         prefer.showmap = FALSE;
         prefer.changes = TRUE;
                             /* Force Redrawing of menu's               */

      }
      else
```

```
      {
         prefer.showmap = TRUE;
         prefer.changes = TRUE;
                        /* Force Redrawing of menu's                    */
      }
   break;
   case 3:
      current = MO_FirstWindow();
      while (current != NIL)
      {
         MO_HideWindow(current);
         current = MO_NextWindow(current);
      }
      prefer.changes = TRUE;/* Adjust menu bar appropriately            */
   break;
   case 5:
   case 6:
   case 7:
   case 8:
   case 9:
      MO_ShowWindow(theItem - 5);
      prefer.changes = TRUE;/* Adjust menu bar appropriately            */
   break;
   default:
      NumToString(theItem, str);
      MO_Generic( "\pUnimplemented Window Menu item:", str );
   }
}   /* ******************** MO_WindowMenu *************************** */


   /*****************************************************************
   MO_SpecialMenu : miscellany
   ***************************************************************** */
   void
MO_SpecialMenu(theItem)
   int               theItem;
{
Str255               str;
WindowPtr            current;
GrafPtr              savePort;

   switch (theItem)
   {
   case 1:                    /* **** DISPLAY UNITS ****               */
      MO_SetUnits();
      MO_ForceUpdate();       /* and force redraw of whole screen     */
   break;
   case 2:                    /* **** TIME STEP ****                  */
      MO_TimeStep();
```

```
   break;
   case 3:                    /* **** TRACE ORBIT ****                    */
      MO_OrbitTrace();
   break;
   case 4:                    /* **** SHOW/HIDE AXES ****                 */
      if (prefer.showaxes)
      {
         prefer.showaxes = FALSE;
         prefer.changes = TRUE;
                              /* Force Redrawing of menu's                */
      }
      else
      {
         prefer.showaxes = TRUE;
         prefer.changes = TRUE;
                              /* Force Redrawing of menu's                */
      }
      MO_ForceUpdate();       /* and force redraw of whole screen         */
   break;
   default:
      NumToString(theItem, &str);
      MO_Generic( "\pUnimplemented Special Menu item:", str );
   }
}   /* ********************* MO_SpecialMenu ************************** */



   /*****************************************************************
   MO_SetUnits : select a new system of units
   ***************************************************************** */
   void
MO_SetUnits()
{
DialogPtr            myDialog;
int                  itemhit;
int                  dummy;
Handle               metric_button, canon_button, english_button, ok_button;
Rect                 rect;

   myDialog = GetNewDialog(SetUnitsID, NIL, INFRONT);
   GetDItem(myDialog, 3, &dummy, &metric_button, &rect);
                              /* metric units radio button                */
   GetDItem(myDialog, 5, &dummy, &canon_button, &rect);
                              /* canonical units radio button             */
   GetDItem(myDialog, 6, &dummy, &english_button, &rect);
                              /* english units (unsupported) button       */

   SetCtlValue(metric_button, 0);
   SetCtlValue(canon_button, 0);
```

```
    HiliteControl(english_button, 255);
                            /* Disable this radio button                */
    if (prefer.std_units == 0)
       SetCtlValue(metric_button, 1);
    else if (prefer.std_units == 1)
       SetCtlValue(canon_button, 1);
    itemhit = 0;

    GetDItem(myDialog, 1, &dummy, &ok_button, &rect);
    MO_OutlineButton( rect, myDialog );

    while ((itemhit != 1) && (itemhit != 2))
    {
       ModalDialog(NIL, &itemhit);
       if (itemhit == 3)
       {
          SetCtlValue(metric_button, 1);
          SetCtlValue(canon_button, 0);
       }
       else if (itemhit == 5)
       {
          SetCtlValue(metric_button, 0);
          SetCtlValue(canon_button, 1);
       }
    }

    if (itemhit == 1)          /* OK button so determine changes          */
    {
       if (GetCtlValue(metric_button) == 1)
          prefer.std_units = 0;
                            /* The final choice was metric units        */
       else if (GetCtlValue(canon_button) == 1)
          prefer.std_units = 1;
                            /* The final choice was canonical units     */
    }
    DisposDialog(myDialog);
}   /* ********************** MO_SetUnits ************************** */



    /*********************************************************************
    MO_TimeStep : select a new compression ratio or plot increment
    ********************************************************************* */
    void
MO_TimeStep()
{
DialogPtr           myDialog;
int                 itemhit;
int                 dummy;
Handle              compress, increment, ok_button;
```

```
double              comp_val, incr_val;
Rect                rect;
Str255              str;

   myDialog = GetNewDialog(TimeStepID, NIL, INFRONT);
   GetDItem(myDialog, 8, &dummy, &compress, &rect);
                           /* compression ratio text                  */
   GetDItem(myDialog, 6, &dummy, &increment, &rect);
                           /* plotting increment text                 */

   sprintf(str, "%G",prefer.time_comp);
   CtoPstr(&str);
   SetIText(compress, str);
   sprintf(str, "%G",prefer.draw_incr);
   CtoPstr(&str);
   SetIText(increment, str);
   SelIText(myDialog, 6, 0, 99);
                           /* Hilight plotting increment text         */
   itemhit = 0;
   GetDItem(myDialog, 1, &dummy, &ok_button, &rect);
   MO_OutlineButton( rect, myDialog );

   while ((itemhit != 1) && (itemhit != 2))
   {
      ModalDialog(NIL, &itemhit);
      if (itemhit == 1)
      {
         GetIText(compress, &str);
         comp_val = MO_pStr2Num(&str);
         if (comp_val <= 0.0)
            itemhit = 0;     /* invalidate OK button selection         */
         GetIText(increment, &str);
         incr_val = MO_pStr2Num(&str);
         if (incr_val <= 0.0)
            itemhit = 0;     /* invalidate OK button selection         */
      }
   }

   if (itemhit == 1)        /* OK button so set changes                */
   {
      prefer.time_comp = comp_val;
      prefer.draw_incr = incr_val;
   }

   DisposDialog(myDialog);
}   /* ********************** MO_TimeStep *************************** */



   /*****************************************************************
```

```
   MO_OrbitTrace : select a method of plotting orbits
                   0 - dots; 1 - lines; 2 - dots on lines
   ************************************************************** */
   void
MO_OrbitTrace()
{
GrafPtr              savePort;
DialogPtr            myDialog;
int                  itemhit, olditem;
int                  dummy, i;
long int             themethod;
Handle               dots_icon, line_icon, dotline_icon, ok_button;
Rect                 rect;
Rect                 icon_rect[3];
Str255               str;

   GetPort(&savePort);
   myDialog = GetNewDialog(TraceOrbitID, NIL, INFRONT);
   SetPort((GrafPtr) myDialog);

   GetDItem(myDialog, 3, &dummy, &dots_icon, &(icon_rect[0]));
                           /* dots icon                                */
   GetDItem(myDialog, 4, &dummy, &line_icon, &(icon_rect[1]));
                           /* solid line icon                          */
   GetDItem(myDialog, 5, &dummy, &dotline_icon, &(icon_rect[2]));
                           /* dotted line icon                         */

   itemhit = prefer.draw_method + 3;
   themethod = prefer.draw_method;
                           /* icons are items 3,4,5 respectively       */

   for (i=0; i<=2; i=i+1)
      InsetRect(&(icon_rect[i]), -4, -4);

   PenMode(patXor);        /* Erase any existing rect with redraws     */
   PenSize(2,2);

   olditem = itemhit;
   BeginUpdate((WindowPtr)(myDialog));
      DrawDialog(myDialog);
      FrameRect(&(icon_rect[itemhit-3]));
   EndUpdate((WindowPtr)(myDialog));

   GetDItem(myDialog, 1, &dummy, &ok_button, &rect);
   MO_OutlineButton( rect, myDialog );

   while ((itemhit != 1) && (itemhit != 2))
   {
      ModalDialog(NIL, &itemhit);
      if ((3 <= itemhit) && (itemhit <= 5) && (olditem != itemhit))
```

```
      {
         FrameRect(&(icon_rect[olditem-3]));
         FrameRect(&(icon_rect[itemhit-3]));
         themethod = (itemhit-3);
         olditem = itemhit;
      }
   }

   if (itemhit == 1)        /* OK button so set changes                */
   {
      prefer.draw_method = themethod;
   }

   SetPort(savePort);
   DisposDialog(myDialog);
}   /* ********************** MO_OrbitTrace ************************** */
```

```
/*********************************************************************

          FILENAME            :  MacOrbits.pl.c
          DESCRIPTION         :  contains drawing (plotting) routines for
                                 orbit window contents.
          ENVIRONMENT         :  Macintosh SE 1Mb
                                 LightSpeed™ C v2.15
          AUTHOR              :  Captain Kenneth L. BEUTEL USMC
          ADVISORS            :  Prof. Dan Davis
                                 Prof. Dan Boger
                                 Naval Postgraduate School, Monterey CA
          REMARKS             :  called by menu code in MacOrbits.menu.c
          VERSION             :  0.9 (3/6/88)

          CHANGES             :  3/6/88 Formatted for MacWrite conversion
********************************************************************* */

#include "QuickDraw.h"
#include "MacTypes.h"
#include "FontMgr.h"
#include "WindowMgr.h"
#include "MenuMgr.h"
#include "TextEdit.h"
#include "DialogMgr.h"
#include "EventMgr.h"
#include "DeskMgr.h"
#include "FileMgr.h"
#include "ToolboxUtil.h"
#include "ControlMgr.h"
#include "stdio.h"
#include "math.h"

#include "LightSpeed Disk:Thesis C f:StdLib.h"

#include "MacOrbits.h"
#include "MacOrbits.proto.h"

extern  Preferences         prefer;
                            /* preferences for how things look        */
extern  BitMap              globePics[MAXGLOBES];
extern  WindowPtr           mapWindow;
                            /* backgd window of world map             */


   /*********************************************************************
   MO_TextOnly : writes data into window for text only display
   ********************************************************************* */
   void
MO_TextOnly( orbitinfo)
   OrbitInfo        orbitinfo;
```

```
{
static Str30        title1[10] =
                      {"\pSemimajor", "\pEccentricity", "\pInclination",
                       "\pArg. of Perigee", "\pLong. of Ascending Node",
                       "\pMean Anomaly", "\pEpoch Time", "\pEpoch Date",
                       "\pEccentric Anomaly", "\pPeriod" };
static Str30        title2[10] =
                      {"\pPerigee Radius","\pApogee Radius","\pMean Motion",
                       "\pAngular Momentum", "\pSemiparameter",
                       "\pCurrent Radius", "\pSwath Width", "\pCurrent
Velocity",
                       "\pEscape Velocity", "\pSpecific Energy"  };
int                 i;
Rect                infoRect, eraser;
Str255              str;
Dist                a, tempd;
Real                e;
Angle               inc;
Dist                radius; /* the current radius km                    */
Real                velocity;/* the current velocity km/sec             */
Dist                x,y,z;
Angle               eccen_anom;


   TextFace(bold | extend);
   MoveTo( (408/2) - (StringWidth("\pTextOnly View")/2), 15);
   DrawString("\pTextOnly View");
   TextFace(0);
   PenSize(2,2);
   MoveTo(0,175);           /* Draw bottom horizontal line              */
   LineTo(408,175);
   MoveTo(0,22);            /* Draw top horizontal line                 */
   LineTo(408,22);
   MoveTo(408/2, 22);
   LineTo(408/2,175);       /* draw center vertical line                */
   PenSize(1,1);


   for ( i=0; i<10; i=i+1)
   {                        /* Draw 1st column titles                   */
      MoveTo(10, 35+(15*i) );
      DrawString(title1[i]);
   }
   for ( i=0; i<10; i=i+1)
   {                        /* Draw 2nd column titles                   */
      MoveTo((408/2) +10, 35+(15*i) );
      DrawString(title2[i]);
   }

                            /* Get Eccentric Anomaly                    */
   eccen_anom = kl_gecca(&(orbitinfo.orbitdata), orbitinfo.orbitdata.epoch);
```

```
/* *******  Create data for left column  ********                    */
   SetRect(&infoRect,135, 24, 203, 175);
                           /* left, top, right, bottom                */
   FrameRect(&infoRect);
   for ( i=0; i<10; i=i+1)
   {
      MoveTo(infoRect.left + 5, 35+(15*i));
      switch (i)
      {
      case 0:              /* Semi Major axis                         */
         a = gl_gorba(orbitinfo.orbitdata);
         if (prefer.std_units == 0)
         {                 /* metric units                            */
            sprintf(&str, "%-7.1f km", a);
          }
         else if (prefer.std_units == 1)
         {                 /* convert from canonical units            */
            tempd = a / 6378.6;
            sprintf(&str, "%-5.3f E.R.", tempd);
          }
         CtoPstr(&str);
      break;
      case 1:              /* eccentricity                            */
         e = gl_gorbe(orbitinfo.orbitdata);
         sprintf(&str, "%-7.5f", e);
         CtoPstr(&str);
      break;
      case 2:              /* inclination                             */
         inc = cv_gangd(gl_gorbi(orbitinfo.orbitdata));
         sprintf(&str, "%-7.2f°", inc);
         CtoPstr(&str);
      break;
      case 3:              /* argument of perigee                     */
         sprintf(&str, "%-7.2f°",cv_gangd(gl_gorbp(orbitinfo.orbitdata)) );
         CtoPstr(&str);
      break;
      case 4:              /* longitude of ascending node             */
         sprintf(&str, "%-7.2f°",cv_gangd(gl_gorbl(orbitinfo.orbitdata)) );
         CtoPstr(&str);
      break;
      case 5:              /* mean anomaly                            */
         sprintf(&str, "%-7.5f rad",
                 cv_gangr(gl_gorbm(orbitinfo.orbitdata)) );
         CtoPstr(&str);
         SetRect(&eraser, 136, 35+(15*(i-1)), 202, 35+(15*i));
         EraseRect(&eraser);
      break;
      case 6:              /* Current time in secs past epoch         */
         IUTimeString(prefer.time, TRUE, str);
```

```
        SetRect(&eraser, 136, 35+(15*(i-1)), 202, 35+(15*i));
        EraseRect(&eraser);
      break;
      case 7:                 /*epoch date                                */
        sprintf(&str, "%-9.0f",orbitinfo.orbitdata.date);
        CtoPstr(&str);
      break;
      case 8:                 /*Eccentric Anomaly                         */
        sprintf(&str, "%-7.5f rad", eccen_anom );
        CtoPstr(&str);
        SetRect(&eraser, 136, 35+(15*(i-1)), 202, 35+(15*i));
        EraseRect(&eraser);
      break;
      case 9:                 /*Period                                    */
        sprintf(&str, "%-9.2f min", gl_gperd(gl_gmean(a))/60.0 );
        CtoPstr(&str);
      break;
      }
      DrawString(str);
   }

/* *******  Create data for right column           ********               */
   SetRect(&infoRect, (408/2) +137, 24, (408/2) +205, 175);
                          /*left, top, right, bottom                      */
   FrameRect(&infoRect);
   for ( i=0; i<10; i=i+1)
   {
      MoveTo(infoRect.left + 5, 35+(15*i));
      switch (i)
      {
      case 0:                 /* radius of perigee                        */
        tempd = gl_gradp(a,e);
        if (prefer.std_units == 0)
        {                     /* metric units                             */
          sprintf(&str, "%-7.1f km", tempd);
         }
        else if (prefer.std_units == 1)
        {                     /* convert from canonical units             */
          tempd = tempd / 6378.6;
          sprintf(&str, "%-5.3f E.R.", tempd);
         }
        CtoPstr(&str);
      break;
      case 1:                 /* radius of apogee                         */
        tempd = gl_grada(a,e);
        if (prefer.std_units == 0)
        {                     /* metric units                             */
          sprintf(&str, "%-7.1f km", tempd);
         }
        else if (prefer.std_units == 1)
        {                     /* convert from canonical units             */
```

```
      tempd = tempd / 6378.6;
      sprintf(&str, "%-5.3f E.R.", tempd);
    }
   CtoPstr(&str);
break;
case 2:                 /* mean motion                          */
   sprintf(&str, "%-7.5f",gl_gmean(a) );
   CtoPstr(&str);
break;
case 3:                 /* angular momentum                     */
   sprintf(&str, "%-7.1f",gl_gangm(a, e));
   CtoPstr(&str);
break;
case 4:                 /* semiparameter                        */
   tempd = gl_gsemi(a, e);
   if (prefer.std_units == 0)
   {                    /* metric units                         */
      sprintf(&str, "%-7.1f km", tempd);
    }
   else if (prefer.std_units == 1)
   {                    /* convert from canonical units         */
      tempd = tempd / 6378.6;
      sprintf(&str, "%-5.3f E.R.", tempd);
    }
   CtoPstr(&str);
break;
case 5:                 /* current radial distance              */
   radius = gl_grade(a,e,eccen_anom);
   tempd = radius;
   if (prefer.std_units == 0)
   {                    /* metric units                         */
      sprintf(&str, "%-7.1f km", tempd);
    }
   else if (prefer.std_units == 1)
   {                    /* convert from canonical units */
      tempd = tempd / 6378.6;
      sprintf(&str, "%-5.3f E.R.", tempd);
    }
   CtoPstr(&str);
   SetRect(&eraser, 343, 35+(15*(i-1)), 406, 35+(15*i));
   EraseRect(&eraser);
break;
case 6:                 /* swath width                          */
   tempd = gl_ggswi(radius);
   if (prefer.std_units == 0)
   {                    /* metric units                         */
      sprintf(&str, "%-7.1f km", tempd);
    }
   else if (prefer.std_units == 1)
   {                    /* convert from canonical units         */
      tempd = tempd / 6378.6;
```

```
            sprintf(&str, "%-5.3f E.R.", tempd);
          }
        CtoPstr(&str);
        SetRect(&eraser, 343, 35+(15*(i-1)), 406, 35+(15*i));
        EraseRect(&eraser);
      break;
      case 7:                /* current velocity                    */
        velocity = gl_gvelo(radius,a);
        sprintf(&str, "%-7.1f km/s", velocity);
        CtoPstr(&str);
        SetRect(&eraser, 343, 35+(15*(i-1)), 406, 35+(15*i));
        EraseRect(&eraser);
      break;
      case 8:                /*escape velocity                     */
        sprintf(&str, "%-7.1f km/s", gl_gvesc(radius));
        CtoPstr(&str);
        SetRect(&eraser, 343, 35+(15*(i-1)), 406, 35+(15*i));
        EraseRect(&eraser);
      break;
      case 9:                /*specific energy                     */
        sprintf(&str, "%-7.1f", gl_gspen(velocity, radius) );
        CtoPstr(&str);
        SetRect(&eraser, 343, 35+(15*(i-1)), 406, 35+(15*i));
        EraseRect(&eraser);
      break;
      }
      DrawString(str);
   }

/* ******* Create data for bottom section:Coord System Location ***** */
   MoveTo(20, 185);
   DrawString("\pCoord System");

/* Internal time formats are: StdLib = decimal hrs, MacOrbits=seconds */
   for (i=0; i<3; i=i+1)
   {
      switch (i)
      {
      case 0:
         strcpy(str, "IJK");
         x= orbitinfo.ijk.x;
         y= orbitinfo.ijk.y;
         z= orbitinfo.ijk.z;
         if (prefer.std_units == 1)
         {                   /* convert to canonical units          */
            x = x / 6378.6;
            y = y / 6378.6;
            z = z / 6378.6;
          }
      break;
      case 1:
```

```
      strcpy(str, "PQW");
      x= orbitinfo.pqw.x;
      y= orbitinfo.pqw.y;
      z= orbitinfo.pqw.z;            /* should always be zero !    */
      if (prefer.std_units == 1)
      {                  /* convert to canonical units             */
         x = x / 6378.6;
         y = y / 6378.6;
         z = z / 6378.6;
       }
   break;
   case 2:
      strcpy(str, "Geographic");
      x= cv_gangd(orbitinfo.geo.latitude);
      y= cv_gangd(orbitinfo.geo.longitude);
      z= orbitinfo.geo.altitude;
      if (prefer.std_units == 1)
      {                  /* convert to canonical units             */
         z = z / 6378.6;
       }
   break;
   }

   MoveTo(25, 200 + (i*15) );
                        /* print the heading                        */
   CtoPstr(&str);
   DrawString(str);
   SetRect(&eraser, 90-1, 200 -11 + (i*15), 350, 200 + (i*15) );
   EraseRect(&eraser);   /* left, top, right, bottom                */

   MoveTo(90, 200 + (i*15) );
   if (i==2)
      sprintf(&str, "lat=%-7.1f", x);
   else
      sprintf(&str, "x  =%-7.1f", x);
   CtoPstr(&str);
   DrawString(str);

   MoveTo(160, 200 + (i*15) );
   if (i==2)
      sprintf(&str, "lon=%-7.1f", y);
   else
      sprintf(&str, "y  =%-7.1f", y);
   CtoPstr(&str);
   DrawString(str);

   MoveTo(230, 200 + (i*15) );
   if (i==2)
      sprintf(&str, "h  =%-7.1f", z);
   else
      sprintf(&str, "z  =%-7.1f", z);
```

```
      CtoPstr(&str);
      DrawString(str);
   }

}   /* ******************** MO_TextOnly ****************************** */



   /*****************************************************************
   MO_DrawElt : writes orb elems into boxes for MO_DrawALL()
   ***************************************************************** */
   void
MO_DrawElt(row, col, str)
   int               row;
   int               col;
   char              *str;
{
int                  centered;

   centered = StringWidth(str)/2;
   MoveTo( (((col-1)*BOX_H) + (70/2) - centered), (row*BOX_V) -3);
   DrawString(str);

}   /* ********************** MO_DrawElt ***************************** */



   /*****************************************************************
   MO_DrawALL : common display routine for all graphics windows
   ***************************************************************** */
   void
MO_DrawALL( orbitinfo, theWind)
   OrbitInfo         orbitinfo;
   WindowPtr         theWind;
{
Rect                 timeRect;
Rect                 eraser;
Str255               timeStr, str;
Angle                a, mean_anom;
int                  i,j;

   timeRect = theWind->portRect;
   timeRect.left = 5;
   timeRect.bottom = timeRect.bottom - (SBarWidth+1);
   timeRect.top    = timeRect.bottom - (SBarWidth);
   timeRect.right = timeRect.right  - (SBarWidth);
   if (timeRect.right > 90)
      timeRect.right = 90;
   EraseRect(&timeRect);
```

```
MoveTo(timeRect.left + 1, timeRect.bottom - 1);
IUTimeString(prefer.time, TRUE, timeStr);
DrawString(timeStr);

PenSize(2, 2);           /* Draw in heavier lines                  */
for (j=0; j<3; j = j+1)  /* Draw boxes around the orb elts         */
{
   MoveTo(0*BOX_H, j*BOX_V);
   LineTo(6*BOX_H, j*BOX_V);
}
for (i=0; i<7; i = i+1)
{
   MoveTo(i*BOX_H, 0*BOX_V);
   LineTo(i*BOX_H, 2*BOX_V);
}
PenSize(1, 1);           /* Rest to old PenSize                    */
TextSize(9);

MO_DrawElt(1, 1, "\pSemimajor");
a = gl_gorba(orbitinfo.orbitdata);
SetRect(&eraser, 2, BOX_V+2, BOX_H-1, 2*BOX_V-1);
if (prefer.std_units == 0)           /* metric units               */
   sprintf(&str, "%-7.1f km", a );
else if (prefer.std_units == 1)
   sprintf(&str, "%-6.3f E.R.", a/6378.0 );
                         /* canonical units                        */
CtoPstr(&str);
EraseRect(&eraser);
MO_DrawElt(2, 1, (char *)str);

MO_DrawElt(1, 2, "\pEccentricity");
sprintf(&str, "%-7.5f", gl_gorbe(orbitinfo.orbitdata) );
CtoPstr(&str);
MO_DrawElt(2, 2, (char *)str);

MO_DrawElt(1, 3, "\pInclination");
sprintf(&str, "%-7.1f°", cv_gangd(gl_gorbi(orbitinfo.orbitdata)) );
CtoPstr(&str);
MO_DrawElt(2, 3, (char *)str);

MO_DrawElt(1, 4, "\pArg of Perigee");
sprintf(&str, "%-7.1f°", cv_gangd(gl_gorbp(orbitinfo.orbitdata)) );
CtoPstr(&str);
MO_DrawElt(2, 4, (char *)str);

MO_DrawElt(1, 5, "\pLong. of ASN");
sprintf(&str, "%-7.1f°", cv_gangd(gl_gorbl(orbitinfo.orbitdata)) );
CtoPstr(&str);
MO_DrawElt(2, 5, (char *)str);

MO_DrawElt(1, 6, "\pMean Anomaly");
```

```
   mean_anom = cv_gangd( gl_gorbm(orbitinfo.orbitdata)
               + gl_gmean(a) * prefer.time );
   mean_anom = fmod(mean_anom, 360.0);
   sprintf(&str, "%-7.1f°", mean_anom );
   CtoPstr(&str);
   SetRect(&eraser, 5*BOX_H+2, BOX_V+2, 6*BOX_H-1, 2*BOX_V-1);
   EraseRect(&eraser);
   MO_DrawElt(2, 6, (char *)str);

}   /* ********************** MO_DrawALL ************************** */



   /*****************************************************************
   MO_DrawIJK : draws orbit in IJK coord system
   ***************************************************************** */
   void
MO_DrawIJK( orbitinfo, theWind)
   OrbitInfo         orbitinfo;
   WindowPtr         theWind;
{
static Rect          dstRect = { 90, 124, 134, 168 };
Rect                 r;
int                  y_center, z_center;
double               ijk_scale;
double               inplane_radius;

   y_center = 90 + ((84-40)/2);
   z_center = 124 + ((68-24)/2);
   ijk_scale = (44.0/2.0) / 6378.0;
                        /* pixels/km                                  */
                        /* Erase old view first                       */
   CopyBits(&globePics[orbitinfo.lastview], &(theWind->portBits),
         &((globePics[orbitinfo.lastview]).bounds), &dstRect, srcXor, NIL);
   if ((prefer.plotting))
   {
      orbitinfo.lastview = (orbitinfo.lastview+1) % 3;
                        /* Now draw new one                           */
      CopyBits(&globePics[orbitinfo.lastview], &(theWind->portBits),
              &((globePics[orbitinfo.lastview]).bounds),
              &dstRect, srcXor, NIL);
   }

   if (prefer.showaxes)
   {                    /* draw the z axis                            */
      MoveTo(z_center, y_center);
      LineTo(z_center, y_center - 30);
                        /* draw the y axis                            */
      MoveTo(z_center, y_center );
      LineTo(z_center + 30, y_center );
                        /* draw the x axis                            */
```

```
   MoveTo(z_center, y_center );
   LineTo(z_center -21, y_center +26 );
}

inplane_radius = orbitinfo.ijk.y * orbitinfo.ijk.y
               + orbitinfo.ijk.z * orbitinfo.ijk.z;
if ((inplane_radius > (6378.0*6378.0)) || (orbitinfo.ijk.x > 0.0))
   switch (prefer.draw_method)
   {
   case 0:                 /* dots only                                  */
      r.top = y_center - MO_trunc(orbitinfo.ijk.z * ijk_scale);
      r.left = z_center + MO_trunc(orbitinfo.ijk.y * ijk_scale);
      r.bottom = r.top + 3;
      r.right = r.left + 3;
      FillOval(&r, black);
   break;
   case 1:                 /* lines only                                 */
      if (prefer.first_plot != TRUE)
      {
         MoveTo(z_center +MO_trunc(orbitinfo.last_ijk.y *ijk_scale),
                y_center -MO_trunc(orbitinfo.last_ijk.z *ijk_scale));
         LineTo(z_center +MO_trunc(orbitinfo.ijk.y * ijk_scale),
                y_center -MO_trunc(orbitinfo.ijk.z * ijk_scale));
      }
   break;
   case 2:                 /* lines and dots                             */
      if (prefer.first_plot != TRUE)
      {
         MoveTo(z_center +MO_trunc(orbitinfo.last_ijk.y *ijk_scale),
                y_center -MO_trunc(orbitinfo.last_ijk.z *ijk_scale));
         r.top = y_center - MO_trunc(orbitinfo.ijk.z * ijk_scale);
         r.left = z_center + MO_trunc(orbitinfo.ijk.y * ijk_scale);
         LineTo(r.left, r.top);
         r.top = r.top - 2;
                         /* center the dot                             */
         r.bottom = r.top + 3;
         r.left = r.left - 2;
                         /* center the dot                             */
         r.right = r.left + 3;
         FillOval(&r, black);
      }
      else               /* just draw a dot                            */
      {
         r.top = y_center - MO_trunc(orbitinfo.ijk.z * ijk_scale) -2;
         r.bottom = r.top + 3;
         r.left = z_center + MO_trunc(orbitinfo.ijk.y *ijk_scale) -2;
         r.right = r.left + 3;
         FillOval(&r, black);
      }
   break;
   }
```

```
}   /* ********************** MO_DrawIJK ***************************** */


   /*****************************************************************
   MO_DrawPQW : draws orbit in PQW coord system
   ***************************************************************** */
   void
MO_DrawPQW( orbitinfo, theWind)
   OrbitInfo         orbitinfo;
   WindowPtr         theWind;
{
static Rect          globe =
                        { PQW_X-15, PQW_Y-15, PQW_X + 15, PQW_Y + 15 };
Rect                 globe_edge;
Rect                 r;
double               pqw_scale = 15.0/6378.0;
                            /* pixels of radii /km radii              */

   MoveTo(10, 44);
   DrawString("\pPQW Graphic view");
   FillOval(&globe, gray);
   globe_edge = globe;
   InsetRect(&globe_edge, -2, -2);
   PenSize(2, 2);
   FrameOval(&globe_edge);
   PenSize(1, 1);
   if (prefer.showaxes)
   {                        /* draw the y axis                        */
      MoveTo(PQW_Y, PQW_X-50 );
      LineTo(PQW_Y, PQW_X+50 );
                            /* draw the x axis                        */
      MoveTo(PQW_Y-100, PQW_X );
      LineTo(PQW_Y+50,  PQW_X );
   }

   switch (prefer.draw_method)
   {
   case 0:                  /* dots only                              */
      r.top = PQW_X - MO_trunc(orbitinfo.pqw.y * pqw_scale);
      r.left = PQW_Y + MO_trunc(orbitinfo.pqw.x * pqw_scale);
      r.bottom = r.top + 3;
      r.right = r.left + 3;
      FillOval(&r, black);
   break;
   case 1:                  /* lines only                             */
      if (prefer.first_plot != TRUE)
      {
         MoveTo(PQW_Y + MO_trunc(orbitinfo.last_pqw.x * pqw_scale),
```

```
                PQW_X - MO_trunc(orbitinfo.last_pqw.y * pqw_scale) );
            LineTo(PQW_Y + MO_trunc(orbitinfo.pqw.x * pqw_scale),
                PQW_X - MO_trunc(orbitinfo.pqw.y * pqw_scale) );
        }
    break;
    case 2:                 /* lines and dots                          */
        if (prefer.first_plot != TRUE)
        {
            MoveTo(  PQW_Y + MO_trunc(orbitinfo.last_pqw.x * pqw_scale),
                     PQW_X - MO_trunc(orbitinfo.last_pqw.y * pqw_scale) );
            r.top = PQW_X - MO_trunc(orbitinfo.pqw.y * pqw_scale);
            r.left = PQW_Y + MO_trunc(orbitinfo.pqw.x * pqw_scale);
            LineTo(r.left, r.top);
            r.top = r.top - 2; /* center the dot                        */
            r.bottom = r.top + 3;
            r.left = r.left - 2;          /* center the dot             */
            r.right = r.left + 3;
            FillOval(&r, black);
        }
        else                    /* and draw a dot                      */
        {
            r.top = PQW_X - MO_trunc(orbitinfo.pqw.y * pqw_scale) - 2;
            r.bottom = r.top + 3;
            r.left = PQW_Y + MO_trunc(orbitinfo.pqw.x * pqw_scale) - 2;
            r.right = r.left + 3;
            FillOval(&r, black);
        }
    break;
    }

}   /* ********************** MO_DrawPQW ***************************** */



    /*************************************************************************
    MO_DrawGEO : draws orbit in GEO coord system
    ************************************************************************* */
    void
MO_DrawGEO( orbitinfo, theWind)
    OrbitInfo           orbitinfo;
    WindowPtr           theWind;
{
static Rect             mapRect= {60, 10, 60+85, 10+208};
                        /* (160+9)/2=85, 614/2=208                     */
int                     xpos, ypos;
double                  pqw_scale= 208.0/360.0;
                        /* pixels /degree long & lat                   */
double                  lat, lon;
Rect                    r;
int                     i;
```

```
   MoveTo(10, 44);
   DrawString("\pGEO Graphic view");
   FrameRect(&mapRect);

   if (prefer.showaxes)
   {
      for (i= 1; i<12; i= i+1 )
      {                        /* draw lines of longitude                    */
         xpos = i*(208/12) + 11;
         if (i!=6)
           PenPat(gray);
         MoveTo(xpos, 60);
         LineTo(xpos, 60+85-1);
         if (i!=6)
           PenNormal();
      }
      for (i= 0; i<6; i= i+1 )
      {                        /* draw lines of latitude                     */
         if (i!=3)
           PenPat(gray);
         ypos = i*(85/6) + 60;
         MoveTo(10,     ypos);
         LineTo(10+208-1, ypos);
         if (i!=3)
           PenNormal();
       }
       PenNormal();
   }

   lat = cv_gangd(orbitinfo.geo.latitude);
   lon = cv_gangd(orbitinfo.geo.longitude);
   if ( (0.0 <= lon) && (lon <= 180.0))
      xpos = (lon * pqw_scale) + (10 + 208/2);
   else if ((180.0 < lon) && (lon <= 360.0))
      xpos = ((lon-180) * pqw_scale) + 10 ;
   else
      xpos = 0;
   ypos = ((90-lat) * pqw_scale) + 60;

   r.top = ypos -2;
   r.bottom = ypos +2;
   r.left = xpos - 2;
   r.right = xpos + 2;

   FillOval(&r, black);

}   /* ********************** MO_DrawGEO ***************************** */


   /*********************************************************************
```

```
   MO_MaintainPlot : updates plotting in each orbit window
   ************************************************************** */
   void
MO_MaintainPlot( )
{
Rect                contentRect;
WindowPtr           theWind;
OrbitInfo           *orbitinfo;
long int            scaling;/* time compression scaling factor       */
long int            increment;/* incremental time since last look    */
GrafPtr             savePort;
RgnHandle           tempRgn;
Angle               eccen_anom;/* New E, after updating sat pos      */

   scaling = 60*60/prefer.time_comp;
                          /* numticks in minute * compression       */
   increment = ((TickCount() - prefer.elapsed_time)/scaling);
   if ( increment < prefer.draw_incr)
      return;             /* Not enough time elapsed yet            */

   prefer.time = prefer.time + increment*60;
                          /* convert into wall clock seconds        */
   if ((prefer.plot_duration == plONE)
     && (prefer.time > prefer.stop_time))
   {
      SysBeep(2);         /* notify user that we are                */
      prefer.plotting = FALSE;/*    out of time - stop plotting     */
      prefer.changes = TRUE;/* Update the menu items                */
      return;
   }
   prefer.elapsed_time = TickCount();
                          /* reset the timer for next update        */
                          /* Determine Eccentric Anomaly            */


   GetPort(&savePort);    /* ******** UPDATE ALL WINDOWS ********   */
   theWind = MC_FirstWindow();
   while (theWind != NIL)
   {
      SetPort(theWind);
                          /* use content rgn bounding box           */
                          /* to mask out scroll bar areas           */
      tempRgn = ( (WindowPeek) theWind)->contRgn;
      contentRect = (*tempRgn)->rgnBBox;
      contentRect.bottom = (contentRect.bottom - contentRect.top)
                          - SBarWidth;
      contentRect.right = (contentRect.right - contentRect.left)
                          - SBarWidth;
      contentRect.top = 0;
      contentRect.left = 0;
      ClipRect(&contentRect);/* Set it in Global coords             */
```

```
    orbitinfo = (OrbitInfo *) (( (WindowPeek) (theWind))->refCon);

    eccen_anom = kl_gecca(&(orbitinfo->orbitdata),
             (orbitinfo->orbitdata).epoch + prefer.time );
    cs_gijkc(orbitinfo->orbitdata, &(orbitinfo->ijk) );
                          /* get position in IJK Coords              */
    cs_gpqwc(orbitinfo->orbitdata, &(orbitinfo->pqw) );
                          /* get position in PQW Coords              */
    cs_ggeoc(orbitinfo->orbitdata,prefer.time/3600.0,
             &(orbitinfo->geo));
                          /* get position in Geographic Coords       */

    if (prefer.showmap)   /* Add each orbit to Map Background        */
       MO_DrawMapUpdate(*orbitinfo, mapWindow);

    if (orbitinfo->textonly)
       MO_TextOnly(*orbitinfo);
    else
    {
       MO_DrawALL(*orbitinfo, theWind);
       switch (orbitinfo->coordinates)
       {
       case IJK_COORDS:
          MO_DrawIJK(*orbitinfo, theWind);
       break;
       case PQW_COORDS:
          MO_DrawPQW(*orbitinfo, theWind);
       break;
       case GEO_COORDS:
          MO_DrawGEO(*orbitinfo, theWind);
       break;
       }                  /* endswitch                               */
    }                     /* endelse                                 */
    orbitinfo->last_geo.altitude = orbitinfo->geo.altitude;
    orbitinfo->last_geo.latitude = orbitinfo->geo.latitude;
    orbitinfo->last_geo.longitude = orbitinfo->geo.longitude;
    orbitinfo->last_ijk.x = orbitinfo->ijk.x;
    orbitinfo->last_ijk.y = orbitinfo->ijk.y;
    orbitinfo->last_ijk.z = orbitinfo->ijk.z;
    orbitinfo->last_pqw.x = orbitinfo->pqw.x;
    orbitinfo->last_pqw.y = orbitinfo->pqw.y;
    orbitinfo->last_pqw.z = orbitinfo->pqw.z;
    ClipRect(&(screenBits.bounds));
                          /* Reset the clipping rectangle            */
    theWind = MO_NextWindow(theWind);
  }                       /* endwhile                                */
  SetPort(savePort);      /* ****** ENDUPDATE ALL WINDOWS *******    */
  prefer.first_plot = FALSE;/* terminating first plot sequence       */

}   /* ********************* MO_MaintainPlot ************************ */
```

```
/*********************************************************************

          FILENAME          :  MacOrbits.pr.c
          DESCRIPTION       :  printing interface for MacOrbits.c
          ENVIRONMENT       :  Macintosh SE 1Mb
                               LightSpeed™ C v2.15
          AUTHOR            :  Captain Kenneth L. BEUTEL USMC
                               (Portions copyright Think Technologies)
          ADVISORS          :  Prof. Dan Davis
                               Prof. Dan Boger
                               Naval Postgraduate School, Monterey CA
          REMARKS           :  uses screendraw routines in MacOrbits.pl.c
          VERSION           :  0.9 (3/6/88)

          CHANGES           :  3/6/88 Formatted for MacWrite conversion
********************************************************************* */

#include "QuickDraw.h"
#include "MacTypes.h"
#include "FontMgr.h"
#include "WindowMgr.h"
#include "MenuMgr.h"
#include "TextEdit.h"
#include "DialogMgr.h"
#include "EventMgr.h"
#include "DeskMgr.h"
#include "FileMgr.h"
#include "ToolboxUtil.h"
#include "ControlMgr.h"
#include "PrintMgr.h"
#include "stdio.h"

#include "LightSpeed Disk:Thesis C f:StdLib.h"

#include "MacOrbits.h"
#include "MacOrbits.proto.h"

extern Preferences  prefer; /* preferences for how things look        */


/* ****** S P E C I F I C   P R I N T I N G   V A R I A B L E S  *** */
static THPrint      hPrint = NIL;



   /*********************************************************************
   MO_CheckPrintHandle : fetches a print handle if one doesn't exist
   ********************************************************************* */
   void
MO_CheckPrintHandle()
{
```

```
    if (hPrint==NIL)
        PrintDefault(hPrint = (TPrint **) NewHandle( sizeof( TPrint )));
}    /* ******************** MO_CheckPrintHandle ******************** */



    /*****************************************************************
    MO_PageSetUp : call the std page setup dialog
    ***************************************************************** */
    void
MO_PageSetUp()
{
char                errormsg[40];

    PrOpen();
    if (PrError())
    {
        sprintf(errormsg, "Sorry, a printer error = %d", PrError());
        CtoPstr(&errormsg);
        MO_Generic(errormsg, "\poccurred.");
    }
    else
    {
        MO_CheckPrintHandle();
        PrStlDialog(hPrint);
        PrClose();
    }
}    /* ********************** MO_PageSetUp ************************** */



    /*****************************************************************
    MO_PrintOrbitText : print the orbit data
    ***************************************************************** */
    void
MO_PrintOrbitText(theWind)
    WindowPtr           theWind;
{
GrafPtr             savePort;
TPrStatus           prStatus;
int                 copies;
char                errormsg[40];
TPPrPort            printPort;
OrbitInfo           *orbitinfo;

    PrOpen();
    MO_CheckPrintHandle();
    if (PrError())
    {
        sprintf(errormsg, "Sorry, a printer error = %d", PrError());
        CtoPstr(&errormsg);
```

```
      MO_Generic(errormsg, "\poccurred.");
   }
   else if (PrJobDialog(hPrint) != 0)
   {
      MO_Wait();
      GetPort(&savePort);
      printPort = PrOpenDoc(hPrint, NIL, NIL);
      TextFont(geneva);
      TextSize(10);
      PrOpenPage(printPort, NIL);
      orbitinfo = (OrbitInfo *) (( (WindowPeek) (theWind))->refCon);

      if (orbitinfo->textonly)
         MO_TextOnly(*orbitinfo);
      else
      {
         MO_DrawALL(*orbitinfo, (WindowPtr) printPort);
         switch (orbitinfo->coordinates)
         {
         case IJK_COORDS:
           MO_DrawIJK(*orbitinfo, (WindowPtr) printPort);
         break;
         case PQW_COORDS:
           MO_DrawPQW(*orbitinfo, (WindowPtr) printPort);
         break;
         case GEO_COORDS:
           MO_DrawGEO(*orbitinfo, (WindowPtr) printPort);
         break;
         }
      }

      PrClosePage(printPort);
      PrCloseDoc(printPort);

      for (copies=MO_HowMany(); copies>0; copies = copies -1)
      {                    /* image the print and then                */
         PrPicFile( hPrint, NIL, NIL, NIL, &prStatus );
      }
      SetPort(savePort);
   }
   PrClose();
   SetCursor( &arrow );
}   /* ********************* MO_PrintOrbitText ********************** */



   /*****************************************************************
   MO_HowMany : get the number of copies requested
   ***************************************************************** */
   int
MO_HowMany()
```

```
{
   return( ((**hPrint).prJob.bJDocLoop==bDraftLoop) ?
           (**hPrint).prJob.iCopies : 1 );
}   /* ******************** MO_HowMany ***************************** */
```

```
/*****************************************************************

          FILENAME            :  MacOrbits.ut.c
          DESCRIPTION         :  utilities for MacOrbits.c
          ENVIRONMENT         :  Macintosh SE 1Mb
                                 LightSpeed™ C v2.15
          AUTHOR              :  Captain Kenneth L. BEUTEL USMC
          ADVISORS            :  Prof. Dan Davis
                                 Prof. Dan Boger
                                 Naval Postgraduate School, Monterey CA
          REMARKS             :  contains misc interface functions
          VERSION             :  0.9 (3/6/88)

          CHANGES             :  3/6/88 Formatted for MacWrite conversion
***************************************************************** */

#include "QuickDraw.h"
#include "MacTypes.h"
#include "FontMgr.h"
#include "WindowMgr.h"
#include "MenuMgr.h"
#include "TextEdit.h"
#include "DialogMgr.h"
#include "EventMgr.h"
#include "DeskMgr.h"
#include "FileMgr.h"
#include "ToolboxUtil.h"
#include "ControlMgr.h"
#include "stdio.h"

#include "LightSpeed Disk:Thesis C f:StdLib.h"

#include "MacOrbits.h"
#include "MacOrbits.proto.h"

extern  Cursor      watch;
extern  Preferences prefer; /* preferences for how things look        */



   /*****************************************************************
   MO_Wait : display the wait cursor (used for long operations)
   ***************************************************************** */
   void
MO_Wait()
{
   SetCursor( &watch );
}   /* ********************** MO_Wait ***************************** */
```

```
   /*********************************************************************
   MO_pStrCopy : copies one Pascal string from p1 to p2
   ********************************************************************* */
   void
MO_pStrCopy( p1, p2 )
   char              *p1, *p2;
{
register int         len;

   len = *p2++ = *p1++;
   while (--len>=0) *p2++=*p1++;
}   /* ********************* MO_pStrCopy ***************************** */


   /*********************************************************************
   MO_pStrConcat : concatenates pascal string p2 behind p1 and
                   returns ptr to result out.
   ********************************************************************* */
   void
MO_pStrConcat( p1, p2, out )
   Str255            p1, p2, out;
{
register int         i;
int                  len1, len2;
char                 temp[256];

   len1 = p1[0]+1;          /* length including length byte           */
   len2 = p2[0]+1;

   for (i=0; i<len1; i = i+1)
       temp[i] = p1[i];

   for (i=1; i<len2; i = i+1)
       temp[(len1-1) +i] = p2[i];

   temp[0] = len1 + len2 - 2;
   MO_pStrCopy(temp, out);

}   /* ******************** MO_pStrConcat ************************** */


   /*********************************************************************
   MO_pStr2Num : converts pascal string str into a floating point
   ********************************************************************* */
   double
MO_pStr2Num( str )
   Str255            *str;
{
int                  items, next;
```

```
double              after;
double              result;
char                temp[256];

    MO_pStrCopy(str,temp);
    if (temp[0] == 0)
      return(-1.0);

    result =0.0;
    next = 1;
    while ((temp[next] != '.') && (next<=temp[0]))
    {                       /* numbers leading decimal point          */
       if ((temp[next]>='0') && (temp[next]<='9'))
          result = result * 10.0 + (temp[next]-'0');
       else
          return(-1.0);
       next = next + 1;
    }
    if (next<=temp[0])
    {                       /* either done or a decimal pt.           */
       if (temp[next] != '.')
          return(-1.0);
       else                 /* not done process numbers after dec     */
       {
          next = next + 1;
          after = 10.0;     /* tenths is first pos after the dec pt.  */
          while (next<=temp[0])
          {
             if ((temp[next]>='0') && (temp[next]<='9'))
                result = result + ( (temp[next]-'0') / after );
             else
                return(-1.0);
             next = next + 1;
             after = after * 10.0;
                            /* shift over to next decimal position    */
          }
       }
    }

    return( result );

}   /* ******************** MO_pStr2Num ***************************** */


   /*****************************************************************
   MO_pStr2Julian : converts pascal string str into a julian date
   ***************************************************************** */
   double
MO_pStr2Julian( str )
   Str255           *str;
```

```
{
int                 next;
double              month,day,year;
double              result;
char                temp[256];

   MO_pStrCopy(str,temp);
   if (temp[0] == 0)
     return(-1.0);

   month = 0.0;
   day   = 0.0;
   year  = 0.0;
   next  = 1;

   while ((temp[next] != '/') && (next<=temp[0]))
   {                        /* numbers before first slash             */
      if ((temp[next]>='0') && (temp[next]<='9'))
         month = month * 10.0 + (temp[next]-'0');
      else
         return(-1.0);
      next = next + 1;
   }

   if (temp[next] != '/')
      return(-1.0);
   else                     /* not done process numbers after slash   */
      next = next + 1;

   while ((temp[next] != '/') && (next<=temp[0]))
   {                        /* numbers between slashes                */
      if ((temp[next]>='0') && (temp[next]<='9'))
         day = day * 10.0 + (temp[next]-'0');
      else
         return(-1.0);
      next = next + 1;
   }

   if (temp[next] != '/')
      return(-1.0);
   else                     /* notdone process numbers after slash2   */
      next = next + 1;

   while ((temp[next] != '/') && (next<=temp[0]))
   {                        /* numbers after slashes                  */
      if ((temp[next]>='0') && (temp[next]<='9'))
         year = year * 10.0 + (temp[next]-'0');
      else
         return(-1.0);
      next = next + 1;
   }
```

```
    result = tl_gjuld(month, day, year);
    return( result );

}   /* ********************* MO_pStr2Julian ************************* */


    /*********************************************************************
    MO_Generic : 2 Pascal strings displayed in a 1 stage Alert
    ********************************************************************* */
    void
MO_Generic( s1, s2 )
    Str255          s1, s2;
{
    ParamText( s1, s2,"\p", "\p");
    Alert( GenericAlertID, 0L );
}   /* ************************ MO_Generic **************************** */


    /*********************************************************************
    MO_OutlineButton : outlines a button round rect with a heavy line
    ********************************************************************* */
    void
MO_OutlineButton( r, theDialog )
    Rect            r;
    DialogPtr       theDialog;
{
PenState            ps;
Rect                local;
GrafPtr             port;

    GetPort(&port);
    SetPort( (GrafPtr) theDialog);
       local = r;
       GetPenState(&ps);
       PenSize(3,3);
       InsetRect(&local, -4,-4);
       FrameRoundRect(&local, 16, 16);
       SetPenState(&ps);
    SetPort(port);

}   /* ********************* MO_OutlineButton ********************** */


    /*********************************************************************
    MO_Pause : stop everything for x seconds
    ********************************************************************* */
    void
MO_Pause( x )
```

```
   int               x;
{
long int             numTicks, finalTicks;

   numTicks = x * 60;
   Delay(numTicks, &finalTicks);

}   /* ************************ MO_Pause ************************** */



   /*********************************************************************
   MO_trunc : trade a double in for an integer
   ********************************************************************* */
   int
MO_trunc( x )
   double            x;
{
int                  result;

   result = x;              /* convert to integer type                 */
   return( result );
}   /* ************************ MO_trunc ************************** */
```

```
/*****************************************************************

          FILENAME            :  MacOrbits.wm.c
          DESCRIPTION         :  window manager interface for MacOrbits.c
          ENVIRONMENT         :  Macintosh SE 1Mb
                                 LightSpeed™ C v2.15
          AUTHOR              :  Captain Kenneth L. BEUTEL USMC
                                 (Portions Copyright Think Technologies)
          ADVISORS            :  Prof. Dan Davis
                                 Prof. Dan Boger
                                 Naval Postgraduate School, Monterey CA
          REMARKS             :  contains window drawing functions
          VERSION             :  0.9 (3/6/88)

          CHANGES             :  3/6/88 Formatted for MacWrite conversion
***************************************************************** */

#include "QuickDraw.h"
#include "MacTypes.h"
#include "FontMgr.h"
#include "WindowMgr.h"
#include "MenuMgr.h"
#include "TextEdit.h"
#include "DialogMgr.h"
#include "EventMgr.h"
#include "DeskMgr.h"
#include "FileMgr.h"
#include "ToolboxUtil.h"
#include "ControlMgr.h"
#include "stdio.h"

#include "LightSpeed Disk:Thesis C ƒ:StdLib.h"

#include "MacOrbits.h"
#include "MacOrbits.proto.h"

extern Preferences  prefer; /* preferences for how things look        */
extern BitMap       globePics[MAXGLOBES];
extern MenuHandle   myMenus[specialM + 1];



Point               theOrigin;/* position to start drawing windows    */
static WindowPtr    WindowStorage[MAXWINDOWS] =
                           { NIL, NIL, NIL, NIL, NIL};


   /*****************************************************************
   MO_CreateWindow : Create the std window used in program
   ***************************************************************** */
   void
```

```
MO_CreateWindow(theWind, slotnum)
   WindowPtr         *theWind;
   int               *slotnum;
{
WindowPeek           myPeek;
OrbitInfo            *orbitinfo;
int                  i,j;
Str255               buildStr;

/*  The array WindowStorage holds MAXWINDOWS worth of window pointers.
    once these are used up this function returns NIL to indicate out of
    memory condition. WindowStorage slots may be reused in window has
    been deallocated with MO_RemoveWindow().                              */

   i=0;
   while ((i<MAXWINDOWS) && (WindowStorage[i] != NIL))
   {                         /* scan for an available slot               */
      i=i+1;
   }

   if ((WindowStorage[i] != NIL) || (i==MAXWINDOWS))
   {
      *theWind = NIL ;       /* no available slots                       */
      return;
   }

   WindowStorage[i] = GetNewWindow( windowID, NULL, INFRONT );
   SetPort(WindowStorage[i]);/* allocate window record on heap           */
                             /* create a ptr for orbit stuff and
                                store initial values in it               */
   TextSize(9);              /* Use 9 pt text                            */
   orbitinfo = (OrbitInfo*) NewPtr( sizeof(OrbitInfo) );

   orbitinfo->slotnum     = i;/* save the slotnumber for later use       */
   orbitinfo->dirty       = FALSE;
   orbitinfo->newfile     = TRUE;
                             /* Assume data source is not a file         */
   orbitinfo->textonly    = FALSE;
   orbitinfo->coordinates = IJK_COORDS;
   orbitinfo->lastview    = 0;/* Current view of spinning globe          */

   NumToString(i, buildStr);
   MO_pStrConcat("\pWindowStorage#", buildStr, buildStr);
   MO_pStrCopy( buildStr, orbitinfo->test );

   myPeek = (WindowPeek) (WindowStorage[i]);
                             /* copy ptr to orbit stuff in window        */
   myPeek->refCon = (long) orbitinfo;
                             /* return the new WindowPtr and slot no     */
   *theWind = WindowStorage[i];
   *slotnum = i;
```

```
   prefer.changes = TRUE;   /* Flag potential changes to menu          */

/* Shift the window origin down and bring up bottom edge in std way   */
   MoveWindow(*theWind, WINDOW_H+(i*5), WINDOW_V+(i*16), FALSE);
   SizeWindow(*theWind,(WH_MAX-WINDOW_H),((296-WINDOW_V)-(i*16)), TRUE);

   ClipRect(&(screenBits.bounds));
                            /* avoid bug from Apple TN #59            */

}   /* ********************* MO_CreateWindow ************************ */



   /*********************************************************************
   MO_isNewFile : Check to see if window's data didn't come from a file
   ********************************************************************* */
   int
MO_isNewFile(theWind)
   WindowPtr         theWind;
{
OrbitInfo            *orbitinfo;

   if ((theWind==NIL) || (!MO_ours(theWind)) )
      return(FALSE);/* no window so can't come from file               */

   orbitinfo = (OrbitInfo*) (( (WindowPeek) theWind)->refCon) ;
   return(orbitinfo->newfile);

}   /* ********************* MO_isNewFile **************************** */



   /*********************************************************************
   MO_isDirty : Check to see if data assoc w/window has been changed
   ********************************************************************* */
   int
MO_isDirty(theWind)
   WindowPtr         theWind;
{
OrbitInfo            *orbitinfo;

   if ((theWind==NIL) || (!MO_ours(theWind)) )
      return(FALSE);/* no window there so it must be clean             */

   orbitinfo = (OrbitInfo*) (( (WindowPeek) theWind)->refCon) ;
   return(orbitinfo->dirty);

}   /* ********************* MO_isDirty ****************************** */
```

```
   /*****************************************************************
   MO_SetDirty : Set dirty bit to indicate window data has been changed
   ***************************************************************** */
   void
MO_SetDirty(theWind, thebit)
   WindowPtr         theWind;
   Boolean           thebit;
{
OrbitInfo            *orbitinfo;

   if ((theWind==NIL) || (!MO_ours(theWind)) )
      return;               /* no window there so it must be clean    */

   orbitinfo = (OrbitInfo*) (( (WindowPeek) theWind)->refCon) ;
   orbitinfo->dirty = thebit;

}   /* ******************** MO_SetDirty ***************************** */



   /*****************************************************************
   MO_ours : See if window is one created by the program
   ***************************************************************** */
   int
MO_ours(theWind)
   WindowPtr         theWind;
{ /* Scans through the Window storage list tying to match pointers    */
int                  i;

   i = 0;
   while ((i < MAXWINDOWS) && (theWind != WindowStorage[i]))
   {
      i = i + 1;
   }

   if ((WindowStorage[i] == NIL) || (i == MAXWINDOWS))
      return(FALSE);/* No match so not the programs window           */
   else
      return(TRUE);          /* It is the programs window             */

}   /* ********************** MO_ours ******************************** */



   /*****************************************************************
   MO_AvailWind : Check to see if there is room for another orbit window
   ***************************************************************** */
   Boolean
MO_AvailWind()
{
```

```
int                 i;

   i = 0;
   while (i < MAXWINDOWS)   /* Scan for any available opening...      */
   {      .
      if (WindowStorage[i] == NIL )
         return(TRUE);      /* if an opening is found return true     */
      i = i + 1;
   }
   return(FALSE);           /* if no opening then return false        */

}   /* ********************* MO_AvailWind ************************** */


   /*************************************************************************
   MO_DuplicWind : Does a window already exist by the same name?
   ************************************************************************* */
   Boolean
MO_DuplicWind(newname)
   Str255           newname;
{
int                 i;
Str255              windname;
Str255              str;

   i = 0;
   while (i < MAXWINDOWS)   /* Scan for any existing window...        */
   {
      if (WindowStorage[i] != NIL )
      {
         GetWTitle(WindowStorage[i], windname);
         if (EqualString(windname, newname, FALSE, FALSE))
         {
            PtoCstr(&windname);
            sprintf(str, "Sorry, the name '%s' is already in use. ",
                    windname);
            CtoPstr(&str);
            MO_Generic(str, "\p Please choose another.");
            return(TRUE);   /* if the name is found return true       */
         }
      }
      i = i + 1;
   }
   return(FALSE);           /* if no opening then return false        */

}   /* ********************* MO_DuplicWind ************************** */


   /*************************************************************************
```

```
   MO_RemoveWindow : Release storage for window created by the program
   ********************************************************************* */
   void
MO_RemoveWindow(theWind)
   WindowPtr         *theWind;
{
WindowPeek           myPeek;
int                  i;
Str255               stri;

   i = 0;
   while ((i < MAXWINDOWS) && (WindowStorage[i] != *theWind) )
   {
      i = i + 1;            /* scan until the pointer is found        */
   }

   if (WindowStorage[i] == *theWind)
      WindowStorage[i] = NIL;/* zap the window from storage           */
   else
      return;

   NumToString(i+1, stri);
   MO_pStrConcat("\pOrbit Window ", stri, &stri);
   SetItem(myMenus[windowM], i + 5, stri);
   DisableItem(myMenus[windowM], i + 5);

   myPeek = (WindowPeek) *theWind;
   DisposPtr(myPeek->refCon);/* deallocate ptr for orbit stuff        */
   HideWindow(*theWind);
   prefer.changes = TRUE;   /* Flag potential changes to menu         */

}   /* ********************* MO_RemoveWindow ************************ */



   /*********************************************************************
   MO_HideWindow : hide the window from view
   ********************************************************************* */
   void
MO_HideWindow(theWind)
   WindowPtr         theWind;
{
int                  i;

   i = 0;
   while ((i < MAXWINDOWS) && (WindowStorage[i] != theWind) )
   {
      i = i + 1;            /* scan until the pointer is found        */
   }

   if (WindowStorage[i] != theWind)
```

```
      return;

   HideWindow(theWind);     /* hide the window                          */
   prefer.changes = TRUE;   /* Flag potential changes to menu           */

}   /* ********************** MO_HideWindow ************************** */



   /*********************************************************************
   MO_ShowWindow : show the window again
   ********************************************************************* */
   void
MO_ShowWindow(slotnum)
   int              slotnum;
{
   if (WindowStorage[slotnum] == NIL)
      return;

   ShowWindow(WindowStorage[slotnum]);
   SelectWindow(WindowStorage[slotnum]);
   prefer.changes = TRUE;   /* Flag potential changes to menu           */

}   /* ********************** MO_ShowWindow ************************** */



   /*********************************************************************
   MO_FirstWindow : Find any pointer to a window created by the program
   ********************************************************************* */
   WindowPtr
MO_FirstWindow()
{ /* Scans through the Window storage list for 1st non empty slot     */
int                 i;

   i = 0;

   while ((i < MAXWINDOWS) && (WindowStorage[i] == NIL))
   {
      i = i + 1;
   }

   if ((WindowStorage[i] == NIL) || (i == MAXWINDOWS))
      return(NIL);          /* No match so no program windows left     */
   else
      return(WindowStorage[i]);
                            /* The first program  window found         */
}   /* ********************** MO_FirstWindow ************************** */
```

```
   /*****************************************************************
   MO_UpdateWindow : Handle update event for program windows
   ***************************************************************** */
   void
MO_UpdateWindow(theWind)
   WindowPtr         theWind;
{
GrafPtr              savePort;
Rect                 r;
Rect                 plotRect;
OrbitInfo            *orbitinfo;
RgnHandle            oldContent;

   GetPort( &savePort );
   SetPort( theWind );
   oldContent =((WindowPeek) theWind)->contRgn;
   r = (*oldContent)->rgnBBox;/* Get content region bounding box      */
   r.bottom = (r.bottom - r.top) - (SBarWidth);
   r.right = (r.right-r.left) - (SBarWidth);
   r.top = 0;
   r.left = 0;
   ClipRect(&r);            /* Clip the scroll bar areas             */

   BeginUpdate( theWind );

   orbitinfo = (OrbitInfo *) (( (WindowPeek) (theWind))->refCon);
   cs_gijkc(orbitinfo->orbitdata, &(orbitinfo->ijk) );
                            /* get position in IJK Coords            */
   cs_gpqwc(orbitinfo->orbitdata, &(orbitinfo->pqw) );
                            /* get position in PQW Coords            */
   cs_ggeoc(orbitinfo->orbitdata,prefer.time/3600.0, &(orbitinfo->geo));
                            /* get position in Geographic Coords*/

   if (orbitinfo->textonly)
   {
      EraseRect(&(theWind->portRect));
      MO_TextOnly(*orbitinfo);
   }
   else
   {
      plotRect = theWind->portRect;
      plotRect.top = plotRect.top + (2*BOX_V+2);
      EraseRect(&plotRect);
      MO_DrawALL(*orbitinfo, theWind);
      switch (orbitinfo->coordinates)
      {
      case IJK_COORDS:
         MO_DrawIJK(*orbitinfo, theWind);
      break;
      case PQW_COORDS:
```

```
            MO_DrawPQW(*orbitinfo, theWind);
         break;
         case GEO_COORDS:
            MO_DrawGEO(*orbitinfo, theWind);
         break;
         }
      }

      ClipRect(&screenBits.bounds);
                              /* Restore clip to whole screen          */
      DrawGrowIcon( theWind );

      EndUpdate( theWind );
      SetPort( savePort );
}     /* ********************* MO_UpdateWindow ************************ */



      /*********************************************************************
      MO_GrowWindow : Handle growWindow event for program windows
      ********************************************************************* */
      void
MO_GrowWindow( theWind, p )
      WindowPtr         theWind;
      Point             p;
{
GrafPtr               savePort;
Rect                  newRect, growRect;
long                  theResult;
Point                 thePt;
Boolean               saveprefer;
Point                 largest;/* largest area that is not erased       */

      GetPort( &savePort );
      SetPort( theWind );
      SetRect(&growRect, 80, 80, WH_MAX-WINDOW_H, screenBits.bounds.bottom);
      theResult = GrowWindow( theWind, p, &growRect );
      thePt.h = LoWord( theResult );
      thePt.v = HiWord( theResult );
                              /* The next 3 calls: SizeWindow(),       */
                              /* EraseRect(), InvalRect()              */
                              /* must come in order specified!         */
                              /* (Chernikoff Vol.2 Pp.103)             */
      newRect = ((*theWind).portRect);
      largest.h = newRect.right ;
      largest.v = newRect.bottom ;

      SizeWindow( theWind, thePt.h, thePt.v, TRUE );
      newRect = ((*theWind).portRect);
      if (newRect.right>largest.h)
```

```
      newRect.left = largest.h- (SBarWidth+1);
   else
   {
      newRect.left = newRect.right- (SBarWidth+1);
      newRect.right = largest.h-1;
   }
   EraseRect( &newRect );   /* only erase increased rect to right     */
   newRect = ((*theWind).portRect);
   if (newRect.bottom>largest.v)
      newRect.top = largest.v- (SBarWidth+1);
   else
   {
      newRect.top = newRect.bottom- (SBarWidth+1);
      newRect.bottom = largest.v-1;
   }
   EraseRect( &newRect );   /* only erase increased rect to bottom    */
   InvalRect( &((*theWind).portRect) );
                            /* Save plotting preferences              */
   saveprefer = prefer.plotting;
   prefer.plotting = FALSE; /* Force a single redraw                  */
   MO_UpdateWindow( theWind );
   prefer.plotting = saveprefer;
                            /* And restore prefernces afterwards      */
   SetPort( savePort );

}   /* ********************* MO_GrowWindow ************************** */



   /*********************************************************************
   MO_ForceUpdate : Creates update events for all open orbit windows
   ********************************************************************* */
   void
MO_ForceUpdate()
{
WindowPtr           current;
GrafPtr             savePort;

      GetPort(&savePort);   /* and force redraw of whole screen       */
      current = MO_FirstWindow();
      while (current != NIL)
      {
          SetPort(current);
          InvalRect(&(current->portRect));
          current = MO_NextWindow(current);
      }
      SetPort(savePort);
}   /* ********************* MO_ForceUpdate ************************** */
```

```
   /*********************************************************************
   MO_NextWindow : Finds pointer to next orbit window in ordered list
   ********************************************************************* */
   WindowPtr
MO_NextWindow(current)
   WindowPtr         current;
{
int                  i;

   i = 0;
   while ((i < MAXWINDOWS) && (WindowStorage[i] != current))
   {                          /* Scans list for current window          */
      i = i + 1;
   }

                              /* Find next non empty slot               */
   if ((WindowStorage[i] == current) && (i == (MAXWINDOWS-1)))
      return(NIL);            /* No match so no program windows left    */

   i = i+1;
   while ((i < MAXWINDOWS) && (WindowStorage[i] == NIL))
   {
      i = i + 1;
   }


   if (i == MAXWINDOWS)
      return(NIL);            /* No match so no program windows left    */
   if (WindowStorage[i] == NIL)
      return(NIL);            /* No match so no program windows left    */
   else
      return(WindowStorage[i]);
                              /* The next program  window found         */
}   /* ********************** MO_NextWindow ************************** */
```

```
*********************************************************************
*
*         FILENAME           :  MacOrbits.R
*         DESCRIPTION        :  resouce source file for MacOrbits.c
*         ENVIRONMENT        :  Macintosh SE 1Mb
*                               LightSpeed™ C v2.15
*         AUTHOR             :  Captain Kenneth L. BEUTEL USMC
*         ADVISORS           :  Prof. Dan Davis
*                               Prof. Dan Boger
*                               Naval Postgraduate School, Monterey CA
*         VERSION            :  0.9 (3/6/88)
*         REMARKS            :  Compile this file with RMAKER for use.
*
*         CHANGES            :  3/6/88   Formatted for MacWrite conversion
*
*********************************************************************
*  The output rsrc filename is:
MacOrbits proj.rsrc

*  Include the world map
INCLUDE World.PICT

*  Include the icon symbols
INCLUDE MacOrbits Icons

*  Include the nps logo
INCLUDE NPS Logo

*  Include the Globes w/ meridian lines
INCLUDE Globes.PICT
*************************** MENU rsrc ID's ***************************
Type MENU

     ,128
\14
About MacOrbits...
(-

     ,129
 File
New/N
Open.../O
Close
(-
Save/S
Save As…
Revert
(-
Page Setup…
Print…
(-
```

```
Transfer.../T
Quit/Q

     ,130
 Edit
Undo/Z
(-
Cut/X
Copy/C
Paste/V
Clear

     ,131
 Orbit
Display as text
Display Graphically
(-
IJK Coordinates
PQW Coordinates
Geographic Coords

     ,132
 Plot
Timed Plot...
Start Continuous Plot
(-
Reset Plot
(-
Stop Plotting

     ,133
 Windows
Global Map Background!\12
(-
Hide All Orbits
(-
(Orbit Window 1
(Orbit Window 2
(Orbit Window 3
(Orbit Window 4
(Orbit Window 5

     ,134
 Special
Units of Measurement...
Time Step...
Trace Orbit...
Show Axes!\12


*************************** WIND rsrc ID's ************************
```

```
Type WIND

     ,128
Standard Orbit Window
40 20 336 482
InVisible GoAway
0
0

     ,600
Map Window
22 2 338 510
Visible NoGoAway
 2
 0


*************************** ALRT rsrc ID's ************************
Type ALRT

     ,256
82 133 244 379
256
5555

     ,257
100 105 250 405
257
5555

     ,258
42 27 310 485
258
4444


*************************** DLOG rsrc ID's ************************

TYPE DLOG

     ,512
NewOrbitID
45 65 320 451
Visible NoGoAway
 1
 0
512


     ,513
```

```
SetUnitsID
65 75 252 438
Visible NoGoAway
 1
 0
513


     ,514
TimeStepID
65 75 252 442
Visible NoGoAway
 1
 0
514


     ,515
TraceOrbitID
72 118 240 420
Visible NoGoAway
 1
 0
515


     ,516
SetPlotTimeID
61 91 209 408
Visible NoGoAway
 1
 0
516

************************ DITL rsrc ID's for DLOGs and ALRTs ********
TYPE DITL

     ,256
2
*   1
BtnItem Enabled
128 96 152 152
OK

*   2
StatText Disabled
17 12 121 236
^0^1^2^3

     ,257
4
```

```
*   1
BtnItem Enabled
67 39 91 125
Save

*   2
BtnItem Enabled
105 39 129 125
Discard

*   3
BtnItem Enabled
105 184 129 270
Cancel

*   4
StatText Disabled
13 68 60 260
Save changes to ^0 ?

********** A b o u t    M a c O r b i t s   DITL
     ,258
10
*   1
BtnItem Enabled
200 288 232 376
OK

*   2
PictItem Disabled
6 8 242 196
258

*   3
StatText Disabled
26 244 41 431
By Capt. K.L. Beutel, USMC

*   4
StatText Disabled
72 224 87 399
In partial fulfilment of :

*   5
StatText Disabled
96 224 112 432
M.S. in Computer Science and

*   6
StatText Disabled
120 224 136 432
```

```
M.S. in Systems Technology at

*   7
StatText Disabled
144 224 160 432
U.S. Naval Postgraduate School,

*   8
StatText Disabled
168 224 184 432
March, 1988.

*   9
StatText Disabled
5 191 21 427
MacOrbits : The Orbital Simulation

*   10
StatText Disabled
240 168 256 440
(Portions Copyright THINK TECHNOLOGIES)

****************** N E W    O R B I T    D I T L
     ,512
28
*   1     OK Button
BtnItem Enabled
232 80 256 166
OK

*   2     Cancel Button
BtnItem Enabled
232 240 256 326
Cancel

*   3     (name)
EditText Enabled
8 112 24 272
Untitled

*   4     (a)
EditText Enabled
32 176 48 240
9378

*   5     (e)
EditText Enabled
56 176 72 240
0.3

*   6
```

```
StatText Disabled
56 16 72 152
Eccentricity (e) :

*   7
StatText Disabled
80 16 96 152
Inclination (i) :

*   8
StatText Disabled
8 16 24 104
Orbit Name :

*   9
StatText Disabled
32 16 48 152
SemiMajor Axis (a) :

*   10    (i)
EditText Enabled
80 176 96 240
45.0

*   11
StatText Disabled
128 16 144 152
Arg. of Perigee :

*   12    (arg of perigee)
EditText Enabled
128 176 144 240
30.0

*   13
StatText Disabled
152 16 168 176
Long. of Ascend. Node :

*   14    (long of ascending node)
EditText Enabled
152 176 168 240
20.0

*   15
StatText Disabled
176 16 192 176
Epoch Time (T) :

*   16    (Epoch Time)
EditText Enabled
```

```
176 176 192 240
10.0

*   17
StatText Disabled
32 256 47 373
(km)

*   18
StatText Disabled
56 256 72 328
(0 ≤ e < 1)

*   19
StatText Disabled
80 256 96 346
(0° ≤ i ≤ 180°)

*   20
StatText Disabled
128 256 144 336
0° - 360°

*   21
StatText Disabled
152 256 168 336
0° - 360°

*   22
StatText Disabled
176 256 192 336
(hours)

*   23
StatText Disabled
200 16 216 104
Epoch Date :

*   24              (Epoch date)
EditText Enabled
200 112 216 241
01/01/1988

*   25
StatText Disabled
200 256 216 359
(mm/dd/yyyy)

*   26
StatText Disabled
104 16 120 152
```

```
Mean Anomaly (M) :

*   27              (Mean Anomaly)
EditText Enabled
104 176 120 240
0.0

*   28
StatText Disabled
104 256 120 336
0° - 360°

***************** S E T   U N I T S   D I T L
     ,513
6
*   1
BtnItem Enabled
136 48 160 134
OK

*   2
BtnItem Enabled
136 216 160 302
Cancel

*   3
radioButton Enabled
40 152 58 292
Metric (MKS)

*   4
StatText Disabled
8 54 24 262
Select a System of Units:

*   5
radioButton Enabled
64 152 82 292
Canonical

*   6
radioButton Disabled
88 152 106 292
English (mi.-lbs.-hr.)


***************** S E T   T I M E   S T E P   D I T L
     ,514
9
*   1
BtnItem Enabled
```

```
144 48 168 134
OK

*   2
BtnItem Enabled
144 216 168 302
Cancel

*   3
StatText Disabled
48 24 64 176
Update position every

*   4
StatText Disabled
16 80 32 288
Change Plotting Parameters

*   5
StatText Disabled
48 224 64 288
Minutes

*   6
EditText Enabled
48 192 64 217
0

*   7
StaticText Disabled
88 24 104 176
Time Compression Ratio

*   8
EditText Enabled
88 186 104 217
0

*   9
StaticText Disabled
88 224 104  280
: 1


***************** T R A C E   O R B I T   D I T L
     ,515
6
*   1
BtnItem Enabled
127 32 151 118
OK
```

```
*   2
BtnItem Enabled
127 176 151 262
Cancel

*   3
IconItem Enabled
60 50 92 82
512

*   4
IconItem Enabled
60 130 92 162
513

*   5
IconItem Enabled
60 200 92 232
514

*   6
StatText Disabled
16 88 32 200
Trace Orbit Path with

****************** S E L E C T   P L O T   D U R A T I O N
     ,516
5
*   1
BtnItem Enabled
96 48 120 134
OK

*   2
BtnItem Enabled
96 184 120 270
Cancel

*   3
StatText Disabled
40 24 56 160
Select Plot Duration

*   4
EditText Enabled
40 176 56 216
90

*   5
StatText Disabled
```

40 232 56 288
Minutes

APPENDIX E

MACORBITS USER'S MANUAL

INTRODUCTION: MacOrbits is a program for the Apple Macintosh computer that simulates the motion of satellites around the Earth. It is designed to be easily used by anyone interested in observing various orbital phenomena such as the motion of a satellite or the trace that the satellite's path makes as it passes over the Earth's surface. This program assumes that the user is familiar with the basic operation of the Macintosh computer and the fundamental principles of orbital mechanics. If you are not comfortable with either of these topics please consult "Macintosh", your owner's manual, or a suitable physics text that describes basic orbit parameters (i.e., semimajor axis, eccentricity, inclination, argument of perigee, longitude of ascending node and epoch time). The thesis that accompanies this program is another source of information about satellite orbits. MacOrbits supplies several default values that may be modified later but (if left unchanged) will help you get started if you are still not entirely comfortable with these topics.

BACKGROUND: This program was written as partial completion of the requirements for a Master's Thesis at the United States Naval Postgraduate School by Captain Kenneth L. Beutel, USMC. This thesis and program are the property of the United States Government. These documents are unclassified, with no limitations on distribution. It is requested that this or similar notices be provided with all copies of this program and associated documentation.

ORGANIZATION: The remainder of this user's manual is composed of a short "get acquainted" section and a reference section. The reference is organized according to the commands that are available via program menus. Menus are composed of several related items that act either globally or in the context of the current, topmost window. Dashed lines are used to group menu items that do similar functions. Many menu items will execute an action simply by selecting them. Other menu items, suffixed by ellipses (...), require more information from the user before they can be successfully executed.

GETTING STARTED: To start using MacOrbits the Macintosh must be turned on and the Finder's desktop visible on the computer screen. Insert the disk containing the program MacOrbits and its associated files. In a few seconds an application icon will appear bearing the name "MacOrbits". Load the program either by "double clicking" the icon or selecting the icon and then choosing "Open" from the File menu. The application will start loading and in a few seconds present a blank screen with a new menu bar.

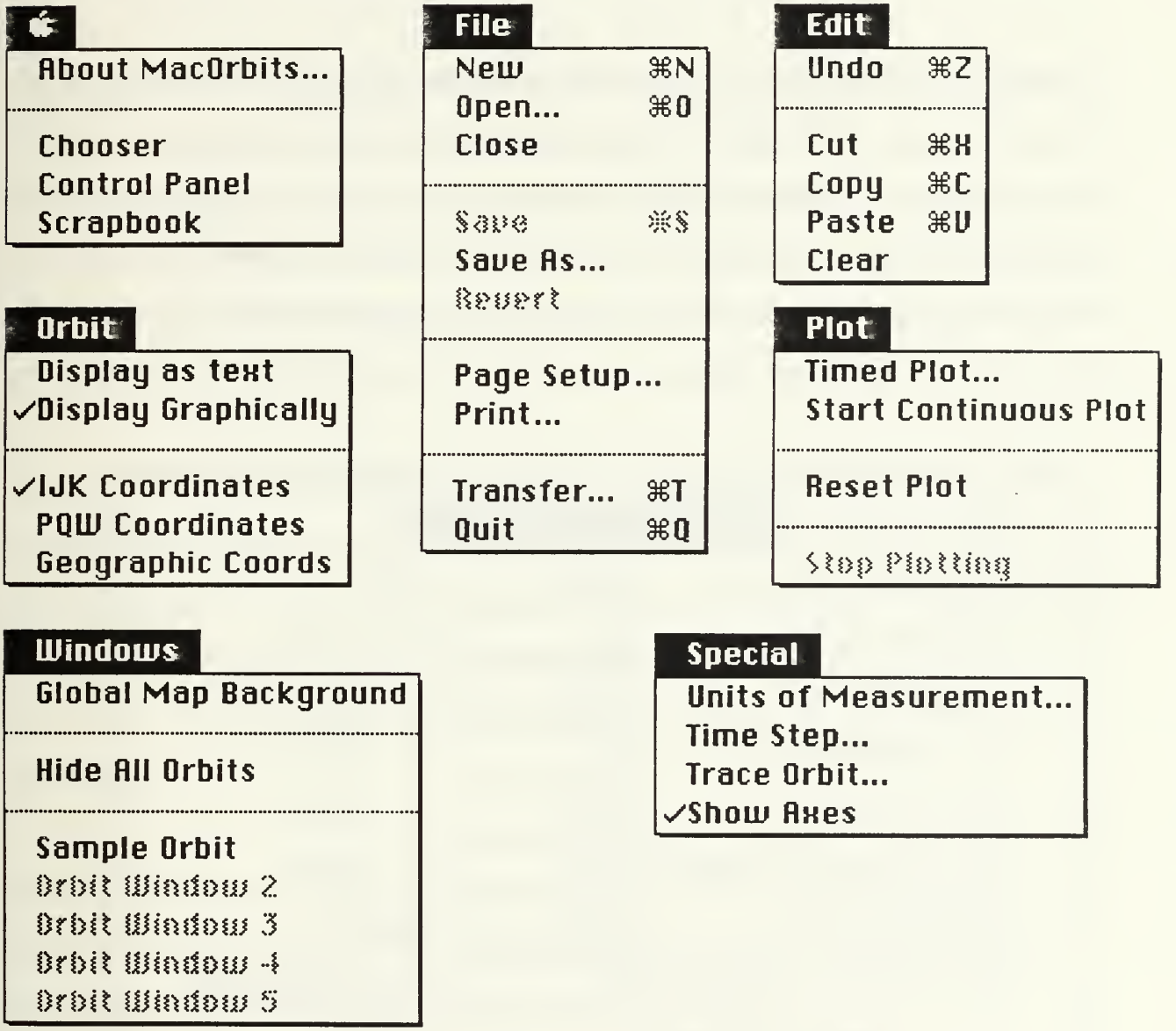
Unless you are already familiar with orbital parameters and would like to define your own orbit, choose "Open" from the File Menu. (If you are familiar with these concepts and want to start making your own orbits immediately, you can skip to the section on Menu Commands.) When the standard file dialog appears, select the name "Demo Orbit" from the file list. After choosing the file and pressing the "Open" button a new window will appear. This window is also named "Demo Orbit" and contains a short list of the orbit parameters that have been defined for this orbit and a globe representing the Earth.

To make the satellite move around the Earth choose "Plot Continuously" from the "Plot" menu and watch the screen. The satellite's position is designated by a black dot in the window. Notice that this black dot and the globe are both updated on a regular basis. To change the rate of this update select the "Time Step..." menu item from the "Special" menu. This dialog will allow you to change the rate at which wall clock time is compressed in the simulation's time system as well as change how frequently new positions are calculated for the satellite. You may experiment with both of these values later, but for now simply double the time compression rate and select the "OK" button. Now the updates are appearing twice as fast in the window.

Notice that you were able to make this change without causing the program to stop plotting or having to do any other special actions. This is because MacOrbits is designed to be as modeless as possible. Except when using dialog boxes, this program allows you to do any action that is legal whenever you want. If a menu item is inactive or dimmed then it is not appropriate to invoke that particular action in the current context. Opening a file, using a desk accessory, or selecting an orbit will activate these menu items as needed and allow you to use them.

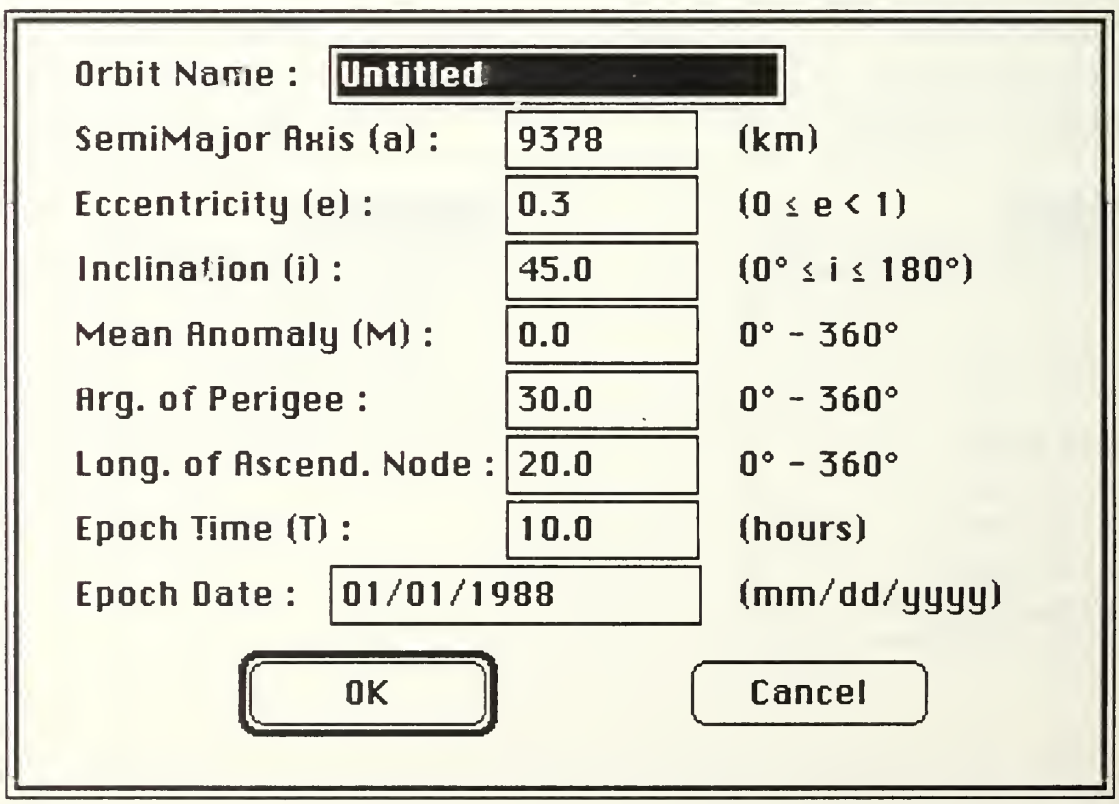
Compatibility Notes: This program has been tested for compatibility with all Macintosh computers from the Macintosh 512KE to the Macintosh II. There is no reason that this program should not run on machines that still have the old 64K (MFS) ROMs or 128K of RAM. Out of memory conditions may develop when using these older machines (or with computers running Switcher™ or MultiFinder™). The most frequent cause of out of memory problems is having too many windows open or using the global map background. Notes are provided in this manual on how to minimize the amount of memory that MacOrbits requires if these problems occur.

REFERENCE: MacOrbits has six menus that contain the following menu items:



 MENU: The Apple Menu contains the currently installed desk accessories and an "About MacOrbits..." item that describes the authorship and purpose of this application.

File MENU: This menu contains the standard Macintosh user interface menu items that allow the user to perform file handling functions, printing functions, and leave the MacOrbits program. Once an orbit file is opened (or created) an orbit window will be displayed with the same title as the file name (or the same title as the new orbit name). MacOrbits will allow up to five orbits to be active at any given time. The "New" menu item creates new orbits by presenting the user with the following dialog box:



The dialog provides a complete set of valid defaults if you are uncertain as to what values to enter to create a working orbit.

The "Open" menu item allows the user to select MacOrbit documents that contain previously written orbit records, organized one per file. The "Close" menu item gives the user the opportunity to save any new orbit data, then closes the file associated with the topmost orbit window, and removes the window and all references to it from the current program. This is not the same as clicking the close box in the window that merely hides the window from view.

The next collection of menu items, "Save", "Save as...", and "Revert" only affect the orbit window that is currently on top of the other orbit windows. The user may explicitly save the topmost orbit at any time by choosing the "Save as..." menu item. "Save" and "Revert" are only active when an existing orbit window has been changed. They allow the user to save changes to disk and return the window to the last saved state of the orbit, respectively.

The "Page Setup" menu item provides the standard Macintosh printer set up routine and allows MacOrbits to support several different printers. The "Print" menu item prints out the values associated with the topmost orbit window.

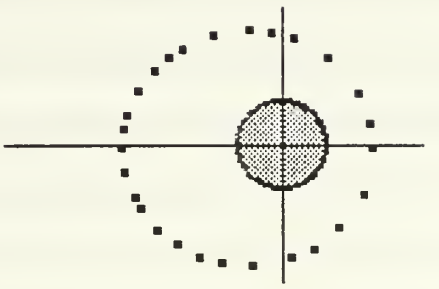
The user has two methods for leaving the MacOrbits application. The "Quit" menu item first ensures that all files have been saved, then exits the program MacOrbits, and returns to the desktop. The "Transfer..." function is similar, except that the user can leave the application and immediately start another application without having to go to the desktop first.

Edit MENU: This menu is not used by the MacOrbits program but is provided for compatibility purposes with desk accessories that require the capability to cut and paste data. The items located in this menu are only active when a desk accessory is the top window on the screen.

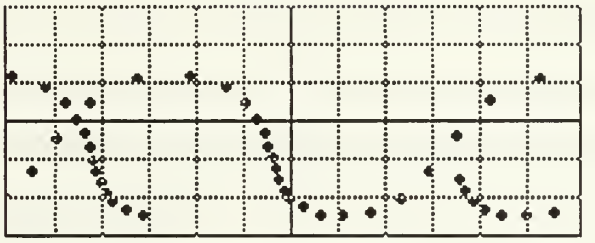
Orbit MENU: This menu is only active when at least one orbit window is opened and visible on the desktop. The Orbit Menu controls the appearance of the topmost orbit window by allowing the user to toggle between a text only view of the orbit and one of three graphical representations of the orbit.

The three possible graphic views are:

IJK Coordinates



PQW Coordinates



Geographic Coordinates

A check mark located in front of a menu item denotes the current active selection. Changing to a new view of the orbit is accomplished by selecting an unmarked item from this menu and then releasing the menu bar. If the selection is made with the "Display Graphically" item check marked, the windows contents will be updated immediately. Otherwise, the selected change will be delayed until the orbit window leaves the "Display as Text" mode.

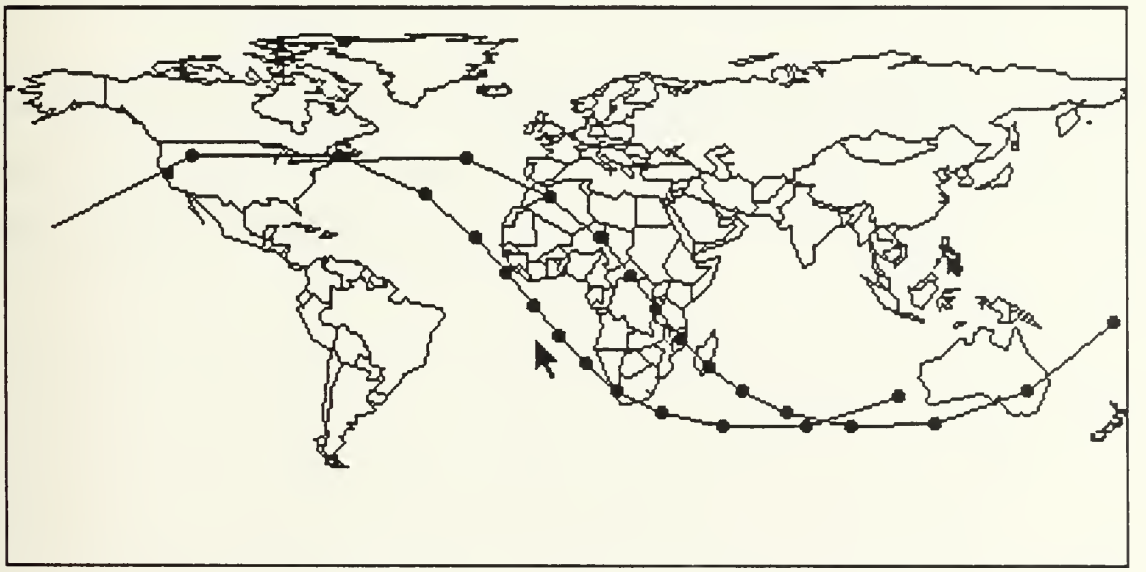
Plot MENU: This menu initiates and terminates the updating of the satellite's orbital position for all open (but not necessarily visible) windows. The user has the option of selecting a continuous plot or plotting for a user-definable interval. Once the program begins plotting the "Stop Plotting" menu item will become active. The user can select this menu item to terminate a plot that was started by either plot option. To reset the orbital parameters to their initial values selct the "Rest Plot" menu item. The rate at which the plotting of the next position will take place is controlled via the "Time Step..." menu item located in the Special Menu.

The plotting mechanism is based on the amount of time elapsed from epoch. Therefore all orbits are each updated once per plot interval. If the "Timed Plot..." time duration expires during that time interval, MacOrbits will beep and reset the menu bar. The "Timed Plot..." menu item uses the following dialog box to determine the duration of the plot from the user:

To plot a single orbit for one active orbit windows: select the desired window, change the view to "Display as Text" via the Orbit Menu, read the time value associated with the period from the data chart, change to the desired plotting view, select the "Timed Plot..." menu item, and set the plot duration to the period of the satellite. The program will then plot the orbit at selected intervals for one complete revolution.

Windows MENU: This menu manages the display of data on the screen. The default desktop for the program is a gray background. This is changeable to a world map that plots the satellites ground position via the "Global Map Background" menu item. (This map consumes about 28K of memory and should not be used under conditions where available memory is at a premium.)

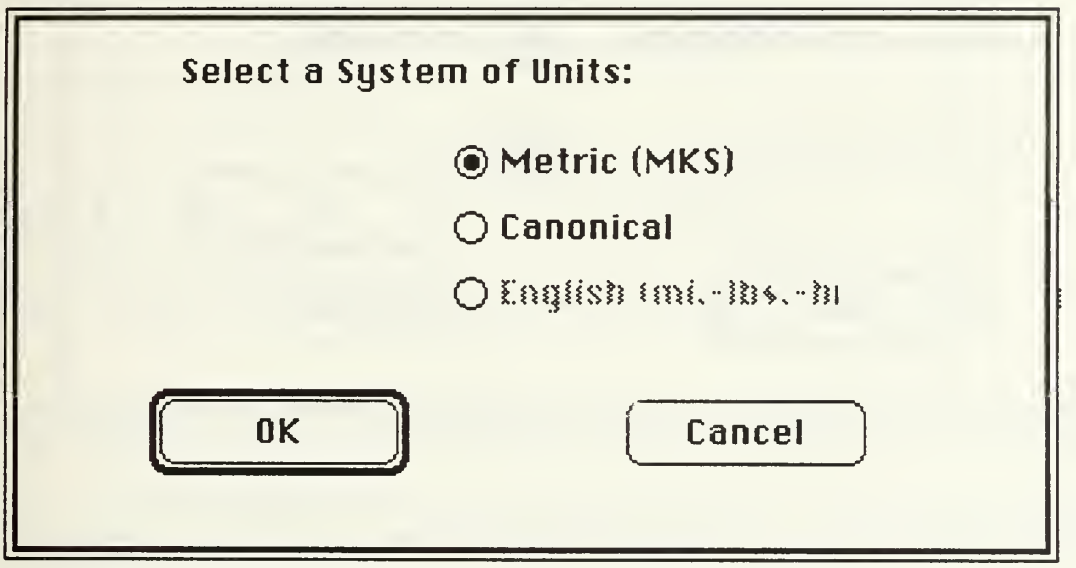
The background window contains a world map with political boundaries that looks like:



The "Hide All Orbits" menu item will remove all orbit windows from the user's view. The orbits associated with each window are not closed but still active and under the control of MacOrbits. This menu item controls the removal of any orbit window that might obscure the view of the global map background.

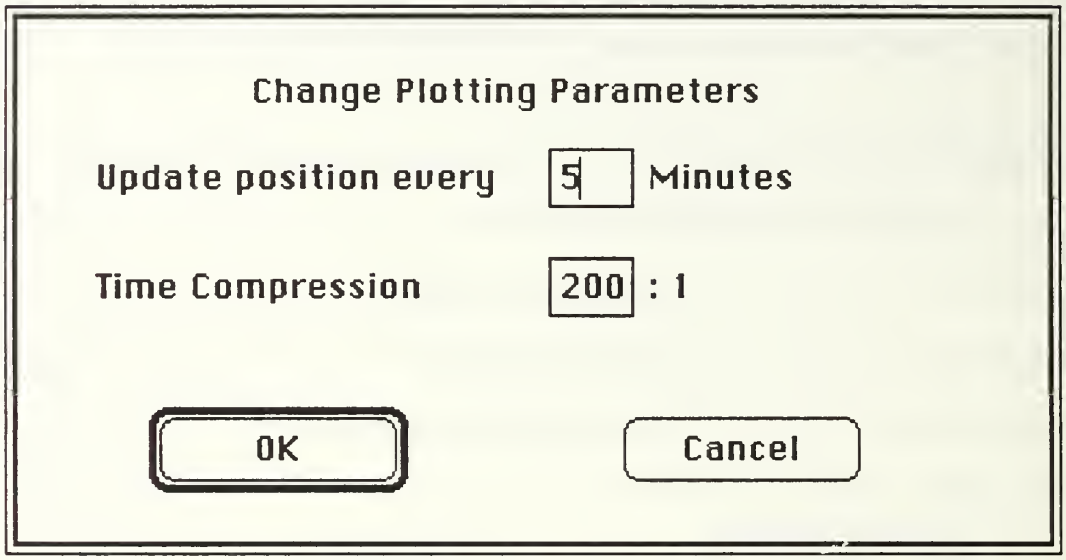
The remaining menu items have the default name of "Orbit Window #" (where # is a number from 1 to 5) and are initially deactivated. One of these five positions will be occupied with the orbit's name when the orbit is created or opened. If a menu item containing the name of an active orbit is selected that orbit's window will be made visible (if it is hidden) and then brought on top of all other visible windows. This allows the user full control over the contents of the display as well as the ability to quickly bring a window from the back of the desktop to the front.

Special MENU: This menu manages changes to the MacOrbits program that are global in scope (i.e., these changes affect all open windows. MacOrbits defaults to metric units (kilometers and seconds) when the program is initially loaded. The "Units of Measurement..." menu item provides a dialog for changing this system of units:



The user may select a new system of units from the choices listed by selecting the button in front of the name of the measurement system. (The third choice is deactivated because English units are not currently supported.)

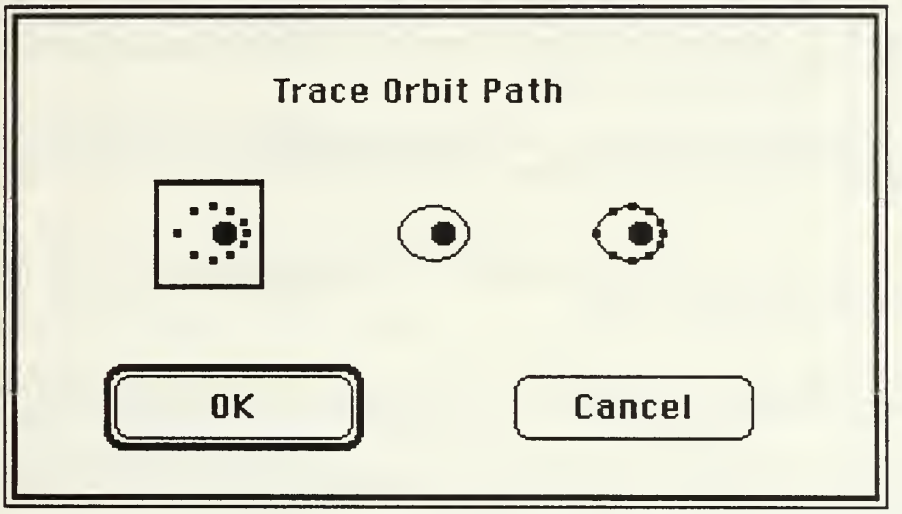
The "Time Step..." menu item controls the rate of time compression (measured versus wall clock time) and the interval between plot points by the dialog:



To change the default values select the appropriate edit text box and type in a new value. Pressing the tab key will highlight the contents of each box sequentially and allows easy selection of a value that the user may wish to type over.

The "Trace Orbit..." menu selection gives the user the option of selecting one of three different methods for drawing the data points at each plot interval. This dialog contains three icons that represent the choice of single points, connecting lines, or both tracing methods combined.

This sample dialog shows the heavy box that indicates the current plot tracing method:



The last menu item, "Show Axes", toggles the display of coordinate axes in all program windows. This option is represented by a check mark found in front of the menu item and initially defaults to active.

REFERENCES

1. Blitzer, L., "A Dynamical Model for Demonstrating Satellite Motions", Sky and Telescope,v. 29, pp. 16-18, January 1965.

2. Bate, R. R., Mueller, D. D., and White, J. E., Fundamentals of Astrodynamics, Dover Publications, Inc., 1971.

3. Air University Publication 18, Space Handbook, Air Command and Staff College, edited by C.D. Cochran, D.M. Gorman, and J.D. Dumoulin, January 1985.

4. Baker, R. M. L. and Makemson, M. W., An Introduction to Astrodynamics, Academic Press Inc., 1967.

5. Naval Research Laboratory Report 7975, Orbital Mechanics of General-Coverage Satellites, by J. A. Eisele and S. A. Nichols, 30 April 1976.

6. Smith, E.A., Orbital Mechanics for Analytic Modeling of Meteorological Satellite Orbits, Atmospheric Science Paper No. 321, Department of Atmospheric Science, Colorado State University, February 1980.

7. Escobal, P. R., Methods of Orbit Determination, John Wiley and Sons, Inc., 1965.

8. Agrawal, B.N., Design of Geosynchronous Spacecraft, Prentice-Hall, Inc., 1986.

9. The World Almanac and Book of Facts, 1974, Newspaper Enterprise Association, 1973.

10. Craig, J. C., 119 Practical Programs for the TRS-80 Pocket Computer, TAB Books, Inc., 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Prof. Rudolf Panholzer, Code 72 Chairman, Space Systems Academic Group Naval Postgraduate School Monterey, California 93943-5000	1
4.	Prof. Vincent Lum, Code 52 Chairman, Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	1
5.	LTC Linda K. Crumback, Code 39 Command, Control and Communications Naval Postgraduate School Monterey, California 93943-5000	1
6.	Computer Technology Curriculum Office Naval Postgraduate School Monterey, California 93943-5000	1
7.	U.S. Space Command Attn: Technical Library Peterson AFB, Colorado 80914	1
8.	Commander Naval Space Command Dahlgren, Virginia 22448	1
9.	Office of the Chief of Naval Operations (OP-943) Department of the Navy Attn: CDR Nelson Spires Washington, D.C. 20350-2000	1
10.	Director, Information Systems (OP-945) Office of the Chief of Naval Operations Department of the Navy Washington, D.C. 20350-2000	2

11. Prof. Daniel L. Davis, Code 52Dv — 1
Computer Science Department
Naval Postgraduate School
Monterey, California 93943-5000

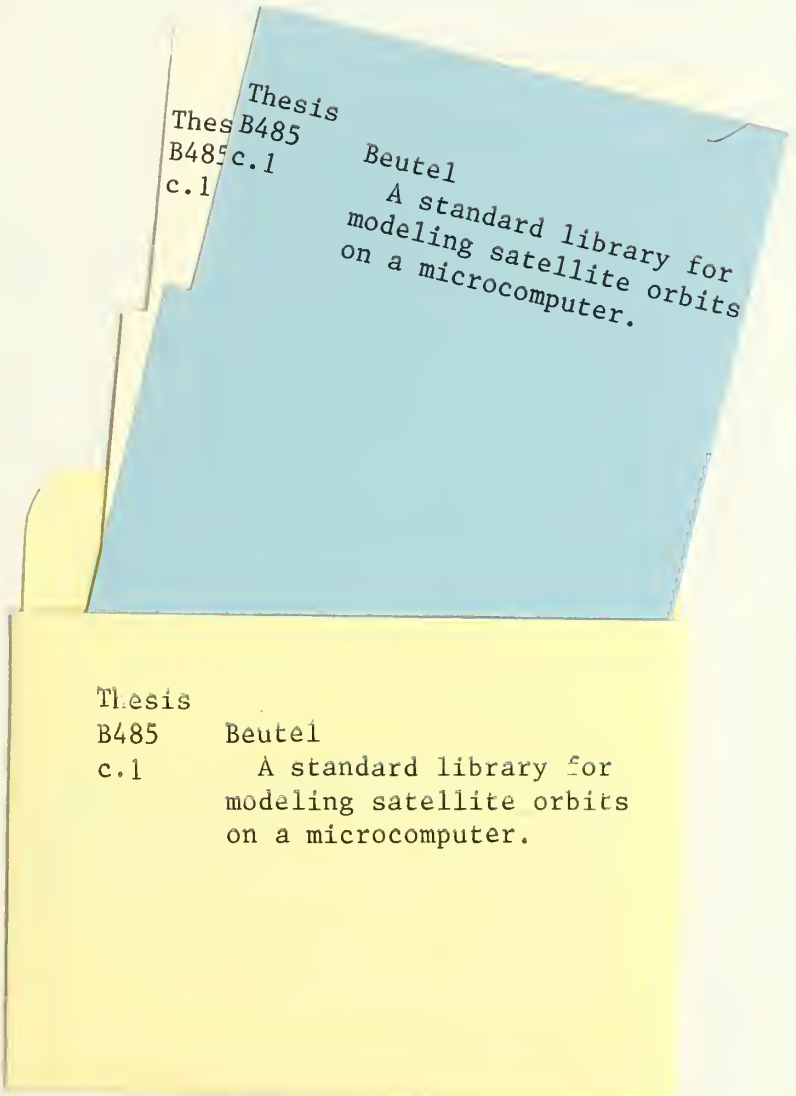
12. Prof. Dan C. Boger, Code 54Bo — 1
Administrative Sciences Department
Naval Postgraduate School
Monterey, California 93943-5000

13. Prof. F. R. Buskirk, Code 61Bs — 1
Physics Department
Naval Postgraduate School
Monterey, California 93943-5000

14. Commander — 1
Pacific Missle Test Center
Attn: Mr. Joseph P. Gannatal
Code 3152
Pt. Mugu, California 93042-5000

15. Capt. William J. Bethke — 1
U.S. Space Command (J3SO) Stop 4
Peterson AFB, Colorado 80914-5001

16. Capt. Kenneth L. Beutel — 1
U.S. Space Command (J3SO) Stop 4
Peterson AFB, Colorado 80914-5001


Thesis
B485
c.1
Beutel
A standard library for modeling satellite orbits on a microcomputer.
Thes
B48
c.1
Thesis
B485
c.1
Beutel
A standard library for modeling satellite orbits on a microcomputer.

thesB485
A standard library for modeling satellit
3 2768 000 78071 2
DUDLEY KNOX LIBRARY