Programming interface to the Swiss Ephemeris

This document describes the proprietary programmer's interface to the Swiss Ephemeris DLL.

1. The programming steps to get a planet's position 5

2. The functions swe_calc_ut() and swe_calc() 7
2.1. The call parameters 7
2.2. Error handling and return values 7
2.3. Bodies (int ipl) 8
Additional asteroids 8
Fictitious planets 11
Obliquity and nutation 13
2.4. Options chosen by flag bits (long iflag) 13
2.4.1. The use of flag bits 13
2.4.2. Ephemeris flags 13
2.4.3. Speed flag 14
2.4.4. Coordinate systems, degrees and radians 14
2.4.5. Specialties (going beyond common interest) 14
a. True or apparent positions 14
b. Topocentric positions 14
c. Heliocentric positions 14
d. Barycentric positions 14
e. Astrometric positions 15
f. True or mean equinox of date 15
g. J2000 positions and positions referred to other equinoxes 15
h. Sidereal positions 15
2.5. Position and Speed (double xx[6]) 15

3. The function swe_get_planet_name() 15

4. Fixed stars functions 16
4.1 swe_fixstar_ut 16
4.2 swe_fixstar() 16
4.3 swe_fixstar_mag() 17

5. Apsides functions 17
5.1 swe_nod_aps_ut 17
5.2 swe_nod_aps() 17

6. Eclipse and planetary phenomena functions 19
6.0. Example of a typical eclipse calculation 19
6.1. swe_sol_eclipse_when_loc() and swe_lun_occult_when_loc() 20
6.2. swe_sol_eclipse_when_glob() 20
6.3. swe_sol_eclipse_how () 21
6.4. swe_sol_eclipse_where () 22
6.5. swe_lun_occult_when_loc() 23
6.6. swe_lun_occult_when_glob() 24

6.7. swe_lun_occult_where () 25
6.8. swe_lun_eclipse_when () 26
6.9. swe_lun_eclipse_how () 26
6.10. swe_rise_trans() and swe_rise_trans_true_hor() (risings, settings, meridian transits) 27
6.11. swe_pheno_ut() and swe_pheno(), planetary phenomena 28
6.12. swe_azalt(), horizontal coordinates, azimuth, altitude 28
6.13. swe_azalt_rev() 29
6.14. swe_refrac(), swe_refract_extended(), refraction 29
6.15. Heliacal risings etc.: swe_heliacal_ut() 30
6.16. Magnitude limit for visibility: swe_vis_limit_mag() 31

7. Date and time conversion functions 31
7.1 Calendar Date and Julian Day: swe_julday(), swe_date_conversion(), /swe_revjul() 31
7.2. UTC and Julian day: swe_utc_time_zone(), swe_utc_to_jd(), swe_jdet_to_utc(), swe_jdut1_to_utc() 32
7.3. Future insertion of leap seconds and the file swe_leapsec.txt 34
7.4. Mean solar time versus True solar time: swe_time_equ() 34

8. Delta T-related functions 34
8.1 swe_deltat() 34
8.2 swe_set_tid_acc(), swe_get_tid_acc() 35
8.3. Future updates of Delta T and the file swe_deltat.txt 35

9. The function swe_set_topo() for topocentric planet positions 35

10. Sidereal mode functions 35
10.1. swe_set_sid_mode() 35
10.2. swe_get_ayanamsa_ut() and swe_get_ayanamsa() 37

11. The Ephemeris file related functions 37
11.1 swe_set_ephe_path() 37
11.2 swe_close() 38
11.3 swe_set_jpl_file() 38
11.4 swe_version() 38

12. House cusp calculation 39
12.1 swe_houses() 39
12.2 swe_houses_armc() 39
12.3 swe_houses_ex() 39

13. The sign of geographical longitudes in Swisseph functions 41

14. Getting the house position of a planet with swe_house_pos() 41

14.1. Calculating the Gauquelin sector position of a planet with swe_house_pos() or swe_gauquelin_sector() 42

15. Sidereal time with swe_sidtime() and swe_sidtime0() 43

16. Summary of SWISSEPH functions 44
16.1. Calculation of planets and stars 44
Planets, moon, asteroids, lunar nodes, apogees, fictitious bodies 44
Fixed stars 44
Set the geographic location for topocentric planet computation 44
Set the sidereal mode for sidereal planet positions 44
16.2 Eclipses and planetary phenomena 45
Find the next eclipse for a given geographic position 45
Find the next eclipse globally 45
Compute the attributes of a solar eclipse for a given tjd, geographic long., latit. and height 45
Find out the geographic position where a central eclipse is central or a non-central one maximal 45
Find the next occultation of a body by the moon for a given geographic position 46
Find the next occultation globally 46
Find the next lunar eclipse 46
Compute the attributes of a lunar eclipse at a given time 46
Compute risings, settings and meridian transits of a body 46
Compute planetary phenomena 47
16.3. Date and time conversion 47
Delta T from Julian day number 47
Julian day number from year, month, day, hour, with check whether date is legal 48
Julian day number from year, month, day, hour 48
Year, month, day, hour from Julian day number 48
Local time to UTC and UTC to local time 48
UTC to jd (TT and UT1) 48

TT (ET1) to UTC 49
UTC to TT (ET1) 49
Get tidal acceleration used in swe_deltat() 49
Set tidal acceleration to be used in swe_deltat() 49
Equation of time 49
16.4. Initialization, setup, and closing functions 49
Set directory path of ephemeris files 49
16.5. House calculation 50
Sidereal time 50
House cusps, ascendant and MC 50
Extended house function; to compute tropical or sidereal positions 50
Get the house position of a celestial point 50
Get the Gauquelin sector position for a body 51
16.6. Auxiliary functions 52
Coordinate transformation, from ecliptic to equator or vice-versa 52
Coordinate transformation of position and speed, from ecliptic to equator or vice-versa 52
Get the name of a planet 52
16.7. Other functions that may be useful 52
Normalize argument into interval [0..DEG360] 52
Distance in centisecs p1 - p2 normalized to [0..360] 52
Distance in degrees 52
Distance in centisecs p1 - p2 normalized to [-180..180] 52
Distance in degrees 53
Round second, but at 29.5959 always down 53
Double to long with rounding, no overflow check 53
Day of week 53
Centiseconds -> time string 53
Centiseconds -> longitude or latitude string 53
Centiseconds -> degrees string 53

17. The SWISSEPH DLLs 53
17.1 DLL Interface for brain damaged compilers 53

18. Using the DLL with Visual Basic 5.0 54

19. Using the DLL with Borland Delphi and C++ Builder 54
19.1 Delphi 2.0 and higher (32-bit) 54
19.2 Borland C++ Builder 55

20. Using the Swiss Ephemeris with Perl 55

21. The C sample program 56

21. The source code distribution 57

22. The PLACALC compatibility API 57

23. Documentation files 58

24. Swisseph with different hardware and compilers 58

25. Debugging and Tracing Swisseph 58
25.1. If you are using the DLL 58
25.2 If you are using the source code 59

Appendix 59
Update and release history 59
Changes from version 1.78 to 1.79 61
Changes from version 1.77 to 1.78 61
Changes from version 1.76 to 1.77 61
Changes from version 1.75 to 1.76 62
Changes from version 1.74 to version 1.75 62
Changes from version 1.73 to version 1.74 62
Changes from version 1.72 to version 1.73 63
Changes from version 1.71 to version 1.72 63
Changes from version 1.70.03 to version 1.71 63
Changes from version 1.70.02 to version 1.70.03 63
Changes from version 1.70.01 to version 1.70.02 63
Changes from version 1.70.00 to version 1.70.01 63
Changes from version 1.67 to version 1.70 63
Changes from version 1.66 to version 1.67 64
Changes from version 1.65 to version 1.66 64
Changes from version 1.64.01 to version 1.65.00 64
Changes from version 1.64 to version 1.64.01 64

Changes from version 1.63 to version 1.64 64
Changes from version 1.62 to version 1.63 64
Changes from version 1.61.03 to version 1.62 65
Changes from version 1.61 to 1.61.01 65
Changes from version 1.60 to 1.61 65
Changes from version 1.51 to 1.60 65
Changes from version 1.50 to 1.51 65
Changes from version 1.40 to 1.50 65
Changes from version 1.31 to 1.40 66
Changes from version 1.30 to 1.31 66
Changes from version 1.27 to 1.30 66
Changes from version 1.26 to 1.27 66
Changes from version 1.25 to 1.26 66
Changes from version 1.22 to 1.23 66
Changes from version 1.21 to 1.22 67
Changes from version 1.20 to 1.21 67
Changes from version 1.11 to 1.20 67
Changes from version 1.10 to 1.11 67
Changes from version 1.04 to 1.10 67
Changes from Version 1.03 to 1.04 67
Changes from Version 1.02 to 1.03 67
Changes from Version 1.01 to 1.02 68
Changes from Version 1.00 to 1.01 68
1. Sidereal time 68
2. Houses 68
3. Ecliptic obliquity and nutation 68

Appendix A 68
What is missing ? 68

Index 69

1. The programming steps to get a planet's position

To compute a celestial body or point with SWISSEPH, you have to do the following steps (use swetest.c as an example). The details of the functions will be explained in the following chapters.

1. Set the directory path of the ephemeris files, e.g.:
   swe_set_ephe_path("C:\\SWEPH\\EPHE");

2. From the birth date, compute the Julian day number:
   jul_day_UT = **swe_julday**(year, month, day, hour, gregflag);

3. Compute a planet or other bodies:
   ret_flag = **swe_calc_ut**(jul_day_UT, planet_no, flag, lon_lat_rad, err_msg);
   or a fixed star:
   ret_flag = **swe_fixstar_ut**(star_nam, jul_day_UT, flag, lon_lat_rad, err_msg);

   Note:
   The functions swe_calc_ut() and swe_fixstar_ut() were introduced with Swisseph version 1.60.
   If you use a Swisseph version older than 1.60 or if you want to work with Ephemeris Time, you have to proceed as follows instead:

   First, if necessary, convert Universal Time (UT) to Ephemeris Time (ET):
   jul_day_ET = jul_day_UT + **swe_deltat**(jul_day_UT);

   Then Compute a planet or other bodies:
   ret_flag = **swe_calc**(jul_day_ET, planet_no, flag, lon_lat_rad, err_msg);
   or a fixed star:
   ret_flag = **swe_fixstar**(star_nam, jul_day_ET, flag, lon_lat_rad, err_msg);

5. At the end of your computations close all files and free memory calling swe_close();

Here is a miniature sample program, it is in the source distribution as swemini.c

```
#include "swephexp.h"  /* this includes  "sweodef.h" */
int main()
{
  char *sp, sdate[AS_MAXCH], snam[40], serr[AS_MAXCH];
  int jday = 1, jmon = 1, jyear = 2000;
  double jut = 0.0;
  double tjd_ut, te, x2[6];
  long iflag, iflgret;
  int p;
  iflag = SEFLG_SPEED;
  while (TRUE) {
    printf("\nDate (d.m.y) ?");
    gets(sdate);
        /* stop if a period . is entered */
    if (*sdate == '.')
      return OK;
    if (sscanf (sdate, "%d%*c%d%*c%d", &jday,&jmon,&jyear) < 1) exit(1);
          /*
           * we have day, month and year and convert to Julian day number
           */
    tjd_ut = swe_julday(jyear,jmon,jday,jut,SE_GREG_CAL);
          /*
           * compute Ephemeris time from Universal time by adding delta_t
           * not required for Swisseph versions smaller than 1.60
           */
      /* te = tjd_ut + swe_deltat(tjd_ut); */
    printf("date: %02d.%02d.%d at 0:00 Universal time\n", jday, jmon, jyear);
    printf("planet     \tlongitude\tlatitude\tdistance\tspeed long.\n");
          /*
           * a loop over all planets
           */
    for (p = SE_SUN; p <= SE_CHIRON; p++) {
      if (p == SE_EARTH) continue;
        /*
```

```
             * do the coordinate calculation for this planet p
             */
        iflgret = swe_calc_ut(tjd_ut, p, iflag, x2, serr);
            /* Swisseph versions older than 1.60 require the following
             * statement instead */
        /* iflgret = swe_calc(te, p, iflag, x2, serr); */
                 /*
                  * if there is a problem, a negative value is returned and an
                  * error message is in serr.
                  */
          if (iflgret < 0)
            printf("error: %s\n", serr);
                 /*
                  * get the name of the planet p
                  */
          swe_get_planet_name(p, snam);
                 /*
                  * print the coordinates
                  */
          printf("%10s\t%11.7f\t%10.7f\t%10.7f\t%10.7f\n",
                snam, x2[0], x2[1], x2[2], x2[3]);
        }
      }
      return OK;
    }
```

2. The functions swe_calc_ut() and swe_calc()

2.1. The call parameters

swe_calc_ut() was introduced with Swisseph **version 1.60** and makes planetary calculations a bit simpler. For the steps required, see the chapter The programming steps to get a planet's position.
swe_calc_ut() and swe_calc() work exactly the same way except that swe_calc() requires Ephemeris Time (more accurate: Dynamical Time) as a parameter whereas swe_calc_ut() expects Universal Time. For common astrological calculations, you will only need swe_calc_ut() and will not have to think anymore about the conversion between Universal Time and Ephemeris Time.
swe_calc_ut() and swe_calc() compute positions of planets, asteroids, lunar nodes and apogees. They are defined as follows:

```
  int swe_calc_ut ( double tjd_ut, int ipl, int iflag, double* xx, char* serr),
where
  tjd_ut =Julian day, Universal Time
  ipl    =body number
  iflag  =a 32 bit integer containing bit flags that indicate what kind of computation is wanted
  xx     =array of 6 doubles for longitude, latitude, distance, speed in long., speed in lat., and speed in dist.
  serr[256] =character string to return error messages in case of error.

and
      int swe_calc(double tjd_et, int ipl, int iflag, double *xx, char *serr),
same but
      tjd_et =    Julian day, Ephemeris time,  where tjd_et = tjd_ut + swe_deltat(tjd_ut)
```

A detailed description of these variables will be given in the following sections.

2.2. Error handling and return values

On success, swe_calc (or swe_calc_ut) returns a 32-bit integer containing flag bits that indicate what kind of computation has been done. This value may or may not be equal to **iflag**. If an option specified by **iflag** cannot be fulfilled or makes no sense, swe_calc just does what can be done. E.g., if you specify that you want JPL ephemeris, but swe_calc cannot find the ephemeris file, it tries to do the computation with any available ephemeris. This will be indicated in the return value of swe_calc. So, to make sure that swe_calc () did exactly what you had wanted, you may want to check whether or not the return code **== iflag**.
However, swe_calc() might return an **fatal error code (< 0)** and an error string in one of the following cases:

- if an illegal body number has been specified
- if a Julian day beyond the ephemeris limits has been specified
- if the length of the ephemeris file is not correct (damaged file)
- on read error, e.g. a file index points to a position beyond file length (data on file are corrupt)
- if the copyright section in the ephemeris file has been destroyed.

If any of these errors occurs,

- the return code of the function is -1,
- the position and speed variables are set to zero,
- the type of error is indicated in the error string **serr.**

2.3. Bodies (int ipl)

To tell **swe_calc()** which celestial body or factor should be computed, a fixed set of body numbers is used. The body numbers are defined in swephexp.h:

```
/* planet numbers for the ipl parameter in swe_calc() */

#define SE_ECL_NUT              -1
#define SE_SUN                  0
#define SE_MOON                 1
#define SE_MERCURY              2
#define SE_VENUS                3
#define SE_MARS                 4
#define SE_JUPITER              5
#define SE_SATURN               6
#define SE_URANUS               7
#define SE_NEPTUNE              8
#define SE_PLUTO                9
#define SE_MEAN_NODE            10
#define SE_TRUE_NODE            11
#define SE_MEAN_APOG            12
#define SE_OSCU_APOG            13
#define SE_EARTH                14
#define SE_CHIRON               15
#define SE_PHOLUS               16
#define SE_CERES                17
#define SE_PALLAS               18
#define SE_JUNO                 19
#define SE_VESTA                20
#define SE_INTP_APOG            21
#define SE_INTP_PERG            22

#define SE_NPLANETS             23
#define SE_FICT_OFFSET          40
#define SE_NFICT_ELEM           15

/* Hamburger or Uranian "planets" */

#define SE_CUPIDO               40
#define SE_HADES                41
#define SE_ZEUS                 42
#define SE_KRONOS               43
#define SE_APOLLON              44
#define SE_ADMETOS              45
#define SE_VULKANUS             46
#define SE_POSEIDON             47

/* other fictitious bodies */

#define SE_ISIS                 48
#define SE_NIBIRU               49
#define SE_HARRINGTON           50
#define SE_NEPTUNE_LEVERRIER    51
#define SE_NEPTUNE_ADAMS        52
#define SE_PLUTO_LOWELL         53
#define SE_PLUTO_PICKERING      54

#define SE_AST_OFFSET           10000
```

Additional asteroids

Body numbers of other asteroids are above SE_AST_OFFSET (=10000) and have to be constructed as follows:
ipl = SE_AST_OFFSET + Minor_Planet_Catalogue_number;

e.g. Eros : ipl = SE_AST_OFFSET + 433
The names of the asteroids and their catalogue numbers can be found in seasnam.txt.

Examples are:

5	Astraea	
6	Hebe	
7	Iris	
8	Flora	
9	Metis	
10	Hygiea	
30	Urania	
42	Isis	not identical with "Isis-Transpluto"
153	Hilda	(has an own asteroid belt at 4 AU)
227	Philosophia	
251	Sophia	
259	Aletheia	
275	Sapientia	
279	Thule	(asteroid close to Jupiter)
375	Ursula	
433	Eros	
763	Cupido	different from Witte's Cupido
944	Hidalgo	
1181	Lilith	(not identical with Dark Moon 'Lilith')
1221	Amor	
1387	Kama	
1388	Aphrodite	
1862	Apollo	(different from Witte's Apollon)
3553	Damocles	highly eccentric orbit betw. Mars and Uranus
3753	Cruithne	("second moon" of earth)
4341	Poseidon	Greek Neptune (different from Witte's Poseidon)
4464	Vulcano	fire god (different from Witte's Vulkanus and intramercurian Vulcan)
5731	Zeus	Greek Jupiter (different from Witte's Zeus)
7066	Nessus	third named Centaur (beween Saturn and Pluto)

There are two ephemeris files for each asteroid (except the main asteroids), a long one and a short one:

se09999.se1	long-term ephemeris of asteroid number 9999, 3000 BC – 3000 AD
se09999s.se1	short ephemeris of asteroid number 9999, 1500 – 2100 AD

The larger file is about 10 times the size of the short ephemeris. If the user does not want an ephemeris for the time before 1500 he might prefer to work with the short files. If so, just copy the files ending with "s.se1" to your hard disk. Swe_calc() tries the long one and on failure automatically takes the short one.
Asteroid ephemerides are looked for in the subdirectories ast0, ast1, ast2 .. ast9 etc of the ephemeris directory and, if not found there, in the ephemeris directory itself. Asteroids with numbers 0 – 999 are expected in directory ast0, those with numbers 1000 – 1999 in directory ast1 etc.

Note that **not all asteroids** can be computed for the whole period of Swiss Ephemeris. The orbits of some of them are extremely sensitive to perturbations by major planets. E.g. **CHIRON**, cannot be computed for the time before **650 AD** and after **4650 AD** because of close encounters with Saturn. Outside this time range, Swiss Ephemeris returns the error code, an error message, and a position value 0. Be aware, that the user will **have to handle** this case in his program. Computing Chiron transits for Jesus or Alexander the Great **will not work.**

The same is true for Pholus before **3850 BC**, and for many other asteroids, as e.g. 1862 Apollo. He becomes chaotic before the year **1870 AD**, when he approaches Venus very closely. Swiss Ephemeris does not provide positions of Apollo for earlier centuries !

Note on asteroid names
Asteroid names are listed in the file seasnam.txt. This file is in the ephemeris directory.

Fictitious planets

Fictitious planets have numbers greater than or equal to 40. The user can define his or her own fictitious planets. The orbital elements of these planets must be written into the file seorbel.txt. The function swe_calc() looks for the file seorbel.txt in the ephemeris path set by swe_set_ephe_path(). If no orbital elements file is found, swe_calc() uses the built-in orbital elements of the above mentioned Uranian planets and some other bodies. The planet number of a fictitious planet is defined as

```
ipl = SE_FICT_OFFSET_1 + number_of_elements_set;
```

e.g. for Kronos: ipl = 39 + 4 = 43.

The file seorbel.txt has the following structure:

```
    # Orbital elements of fictitious planets
    # 27 Jan. 2000
    #
    # This file is part of the Swiss Ephemeris, from Version 1.60 on.
    #
    # Warning! These planets do not exist!
    #
    # The user can add his or her own elements.
    # 960 is the maximum number of fictitious planets.
    #
    # The elements order is as follows:
    # 1. epoch of elements (Julian day)
    # 2. equinox (Julian day or "J1900" or "B1950" or "J2000" or “JDATE”)
    # 3. mean anomaly at epoch
    # 4. semi-axis
    # 5. eccentricity
    # 6. argument of perihelion (ang. distance of perihelion from node)
    # 7. ascending node
    # 8. inclination
    # 9. name of planet
    #
    # use '#' for comments
    # to compute a body with swe_calc(), use planet number
    # ipl = SE_FICT_OFFSET_1 + number_of_elements_set,
    # e.g. number of Kronos is ipl = 39 + 4 = 43
    #
    # Witte/Sieggruen planets, refined by James Neely
J1900, J1900, 163.7409, 40.99837, 0.00460, 171.4333, 129.8325, 1.0833, Cupido   # 1
J1900, J1900,  27.6496, 50.66744, 0.00245, 148.1796, 161.3339, 1.0500, Hades    # 2
J1900, J1900, 165.1232, 59.21436, 0.00120, 299.0440,   0.0000, 0.0000, Zeus     # 3
J1900, J1900, 169.0193, 64.81960, 0.00305, 208.8801,   0.0000, 0.0000, Kronos   # 4
J1900, J1900, 138.0533, 70.29949, 0.00000,   0.0000,   0.0000, 0.0000, Apollon  # 5
J1900, J1900, 351.3350, 73.62765, 0.00000,   0.0000,   0.0000, 0.0000, Admetos  # 6
J1900, J1900,  55.8983, 77.25568, 0.00000,   0.0000,   0.0000, 0.0000, Vulcanus # 7
J1900, J1900, 165.5163, 83.66907, 0.00000,   0.0000,   0.0000, 0.0000, Poseidon # 8
    #
    # Isis-Transpluto; elements from "Die Sterne" 3/1952, p. 70ff.
    # Strubell does not give an equinox. 1945 is taken in order to
    # reproduce the as best as ASTRON ephemeris. (This is a strange
    # choice, though.)
    # The epoch according to Strubell is 1772.76.
    # 1772 is a leap year!
    # The fraction is counted from 1 Jan. 1772
2368547.66, 2431456.5, 0.0, 77.775, 0.3, 0.7, 0, 0, Isis-Transpluto             # 9
    # Nibiru, elements from Christian Woeltge, Hannover
1856113.380954, 1856113.380954, 0.0, 234.8921, 0.981092, 103.966, -44.567, 158.708, Nibiru #
10
    # Harrington, elements from Astronomical Journal 96(4), Oct. 1988
2374696.5, J2000, 0.0, 101.2, 0.411, 208.5, 275.4, 32.4, Harrington             # 11
    # according to W.G. Hoyt, "Planets X and Pluto", Tucson 1980, p. 63
2395662.5, 2395662.5, 34.05, 36.15, 0.10761, 284.75, 0, 0, Leverrier (Neptune)  # 12
2395662.5, 2395662.5, 24.28, 37.25, 0.12062, 299.11, 0, 0, Adams (Neptune)      # 13
2425977.5, 2425977.5, 281, 43.0, 0.202, 204.9, 0, 0, Lowell (Pluto)             # 14
2425977.5, 2425977.5, 48.95, 55.1, 0.31, 280.1, 100, 15, Pickering (Pluto)      # 15
J1900,JDATE, 252.8987988 + 707550.7341 * T, 0.13744, 0.019, 322.212069+1670.056*T,
47.787931-1670.056*T, 7.5, Vulcan # 16
# Selena/White Moon
J2000,JDATE, 242.2205555, 0.05279142865925, 0.0, 0.0, 0.0, 0.0, Selena/White Moon, geo # 17
```

All orbital elements except epoch and equinox may have T terms, where

T = (tjd – epoch) / 36525.
(See, e.g., Vulcan, the second last elements set (not the ”Uranian” Vulcanus but the intramercurian hypothetical planet Vulcan).) ”T * T”, ”T2”, ”T3” are also allowed.
The equinox can either be entered as a Julian day or as ”J1900” or ”B1950” or ”J2000” or, if the equinox of date is required, as ”JDATE”. If you use T terms, note that precession has to be taken into account with JDATE, whereas it has to be neglected with fixed equinoxes.

No T term is required with the mean anomaly, i.e. for the speed of the body, because our software can compute it from semi-axis and gravity. However, a mean anomaly T term had to be added with Vulcan because its speed is not in agreement with the laws of physics. In such cases, the software takes the speed given in the elements and does not compute it internally.

From Version 1.62 on, the software also accepts orbital elements for fictitious bodies that move about the earth. As an example, study the last elements set in the excerpt of seorbel.txt above. After the name of the body, ”, geo” has to be added.

Obliquity and nutation

A special body number SE_ECL_NUT is provided to compute the obliquity of the ecliptic and the nutation. Of course nutation is already added internally to the planetary coordinates by swe_calc() but sometimes it will be needed as a separate value.

```
iflgret = swe_calc(tjd_et, SE_ECL_NUT, 0, x, serr);
```

x is an array of 6 doubles as usual. They will be filled as follows:

```
x[0] = true obliqutiy of the Ecliptic (includes nutation)
x[1] = mean obliquity of the Ecliptic
x[2] = nutation in longitude
x[3] = nutation in obliquity
x[4] = x[5] = 0
```

2.4. Options chosen by flag bits (long iflag)

2.4.1. The use of flag bits

If no bits are set, i.e. if **iflag == 0**, swe_calc() computes what common astrological ephemerides (as available in book shops) supply, i.e. an apparent body position in **geocentric** ecliptic polar coordinates (longitude, latitude, and distance) relative to the true equinox of the date.
If the speed of the body is required, set iflag = SEFLG_SPEED
For mathematical points as the mean lunar node and the mean apogee, there is no apparent position. Swe_calc() returns true positions for these points.
If you need another kind of computation, use the flags explained in the following paragraphs (c.f. swephexp.h). Their names begin with ‚SEFLG_'. To combine them, you have to concatenate them (inclusive-or) as in the following example:

```
iflag = SEFLG_SPEED | SEFLG_TRUEPOS;  (or: iflag = SEFLG_SPEED + SEFLG_TRUEPOS;) // C
iflag = SEFLG_SPEED or SEFLG_TRUEPOS;(or: iflag = SEFLG_SPEED + SEFLG_TRUEPOS;) // Pascal
```

With this value of **iflag**, swe_calc() will compute true positions (i.e. not accounted for light-time) with speed. The flag bits, which are defined in swephexp.h, are:

```
#define SEFLG_JPLEPH        1L          // use JPL ephemeris
#define SEFLG_SWIEPH        2L          // use SWISSEPH ephemeris, default
#define SEFLG_MOSEPH        4L          // use Moshier ephemeris

#define SEFLG_HELCTR        8L          // return heliocentric position
#define SEFLG_TRUEPOS       16L         // return true positions, not apparent
#define SEFLG_J2000         32L         // no precession, i.e. give J2000 equinox
#define SEFLG_NONUT         64L         // no nutation, i.e. mean equinox of date
#define SEFLG_SPEED3        128L        // speed from 3 positions (do not use it, SEFLG_SPEED is
                                        // faster and preciser.)
#define SEFLG_SPEED         256L        // high precision speed (analyt. comp.)
#define SEFLG_NOGDEFL       512L        // turn off gravitational deflection
#define SEFLG_NOABERR       1024L       // turn off 'annual' aberration of light
#define SEFLG_EQUATORIAL    2048L       // equatorial positions are wanted
#define SEFLG_XYZ           4096L       // cartesian, not polar, coordinates
#define SEFLG_RADIANS       8192L       // coordinates in radians, not degrees
#define SEFLG_BARYCTR       16384L      // barycentric positions
#define SEFLG_TOPOCTR       (32*1024L)  // topocentric positions
#define SEFLG_SIDEREAL      (64*1024L)  // sidereal positions
#define SEFLG_ICRS          (128*1024L) // ICRS (DE406 reference frame)
```

2.4.2. Ephemeris flags

The flags to choose an ephemeris are: (s. swephexp.h)

```
SEFLG_JPLEPH        /* use JPL ephemeris */
```

```
SEFLG_SWIEPH        /* use Swiss Ephemeris */
SEFLG_MOSEPH        /* use Moshier ephemeris */
```

If none of this flags is specified, swe_calc() tries to compute the default ephemeris. The default ephemeris is defined in swephexp.h:

```
#define SEFLG_DEFAULTEPH SEFLG_SWIEPH
```

In this case the default ephemeris is Swiss Ephemeris. If you have not specified an ephemeris in **iflag**, swe_calc() tries to compute a Swiss Ephemeris position. If it does not find the required Swiss Ephemeris file either, it computes a Moshier position.

2.4.3. Speed flag

Swe_calc() does not compute speed if you do not add the speed flag SEFLG_SPEED. E.g.

```
iflag |= SEFLG_SPEED;
```

The computation of speed is usually cheap, so you may set this bit by default even if you do not need the speed.

2.4.4. Coordinate systems, degrees and radians

SEFLG_EQUATORIAL	returns equatorial positions: rectascension and declination.
SEFLG_XYZ	returns x, y, z coordinates instead of longitude, latitude, and distance.
SEFLG_RADIANS	returns position in radians, not degrees.

E.g. to compute rectascension and declination, write:

```
iflag = SEFLG_SWIEPH | SEFLG_SPEED | SEFLG_EQUATORIAL;
```

2.4.5. Specialties (going beyond common interest)

a. True or apparent positions

Common ephemerides supply apparent geocentric positions. Since the journey of the light from a planet to the earth takes some time, the planets are never seen where they actually are, but where they were a few minutes or hours before. Astrology uses to work with the positions **we see**. (More precisely: with the positions we would see, if we stood at the center of the earth and could see the sky. Actually, the geographical position of the observer could be of importance as well and topocentric positions could be computed, but this is usually not taken into account in astrology.). The geocentric position for the earth (SE_EARTH) is returned as zero.
To compute the **true** geometrical position of a planet, disregarding light-time, you have to add the flag SEFLG_TRUEPOS.

b. Topocentric positions

To compute topocentric positions, i.e. positions referred to the place of the observer (the birth place) rather than to the center of the earth, do as follows:

- call **swe_set_topo**(geo_lon, geo_lat, altitude_above_sea) (The longitude and latitude must be in degrees, the altitude in meters.)
- add the flag SEFLG_TOPOCTR to **iflag**
- call swe_calc(...)

c. Heliocentric positions

To compute a heliocentric position, add SEFLG_HELCTR.
A heliocentric position can be computed for all planets including the moon. For the sun, lunar nodes and lunar apogees the coordinates are returned as zero; **no error message appears**.

d. Barycentric positions

SEFLG_BARYCTR yields coordinates as referred to the solar system barycenter. However, this option is not completely implemented. It was used for program tests during development. It works only with the JPL and the Swiss Ephemeris, **not with the Moshier** ephemeris; and **only with physical bodies**, but not with the nodes and the apogees.
Moreover, the barycentric Sun of Swiss Ephemeris has ”only” a precision of 0.1”. Higher accuracy would have taken a lot of storage, on the other hand it is not needed for precise geocentric and heliocentric positions. For more precise barycentric positions the JPL ephemeris file should be used.
A barycentric position can be computed for all planets including the sun and moon. For the lunar nodes and lunar apogees the coordinates are returned as zero; no error message appears.

e. Astrometric positions

For astrometric positions, which are sometimes given in the Astronomical Almanac, the light-time correction is computed, but annual aberration and the light-deflection by the sun neglected. This can be done with SEFLG_NOABERR and SEFLG_NOGDEFL. For positions related to the mean equinox of 2000, you must set SEFLG_J2000 and SEFLG_NONUT, as well.

f. True or mean equinox of date

Swe_calc() usually computes the positions as referred to the true equinox of the date (i.e. with nutation). If you want the mean equinox, you can turn nutation off, using the flag bit SEFLG_NONUT.

g. J2000 positions and positions referred to other equinoxes

Swe_calc() usually computes the positions as referred to the equinox of date. SEFLG_J2000 yields data referred to the equinox J2000. For positions referred to other equinoxes, SEFLG_SIDEREAL has to be set and the equinox specified by swe_set_sid_mode(). For more information, read the description of this function.

h. Sidereal positions

To compute sidereal positions, set bit SEFLG_SIDEREAL and use the function swe_set_sid_mode() in order to define the **ayanamsha** you want. For more information, read the description of this function.

2.5. Position and Speed (double xx[6])

swe_calc() returns the coordinates of position and velocity in the following order:

Ecliptic position	**Equatorial position (** SEFLG_EQUATORIAL **)**
Longitude	Rectascension
Latitude	Declination
Distance in AU	distance in AU
Speed in longitude (deg/day)	Speed in rectascension (deg/day)
Speed in latitude (deg/day)	Speed in declination (deg/day)
Speed in distance (AU/day)	Speed in distance (AU/day)

If you need rectangular coordinates (SEFLG_XYZ), swe_calc() returns x, y, z, dx, dy, dz in AU.
Once you have computed a planet, e.g., in ecliptic coordinates, its equatorial position or its rectangular coordinates are available, too. You can get them very cheaply (little CPU time used), calling again swe_calc() with the same parameters, but adding SEFLG_EQUATORIAL or SEFLG_XYZ to **iflag**. swe_calc() will not compute the body again, just return the data specified from internal storage.

3. The function swe_get_planet_name()

This function allows to find a planetary or asteroid name, when the planet number is given. The function definition is

```
char* swe_get_planet_name(int ipl, char *spname);
```

If an asteroid name is wanted, the function does the following:

- The name is first looked for in the asteroid file.
- Because many asteroids, especially the ones with high catalogue numbers, have no names yet (or have only a preliminary designation like 1968 HB), and because the Minor Planet Center of the IAU add new names quite often, it happens that there is no name in the asteroid file although the asteroid has already been given a name. For this, we have the file seasnam.txt, a file that contains a list of all named asteroid and is usually more up to date. If swe_calc() finds a preliminary designation, it looks for a name in this file.

The file seasnam.txt can be updated by the user. To do this, download the names list from the Minor Planet Center http://cfa-www.harvard.edu/iau/lists/MPNames.html, rename it as seasnam.txt and move it into your ephemeris directory.

The file seasnam.txt need not be ordered in any way. There must be one asteroid per line, first its catalogue number, then its name. The asteroid number may or may not be in brackets.

Example:

```
(3192) A'Hearn
(3654) AAS
(8721) AMOS
(3568) ASCII
(2848) ASP
(677) Aaltje
  ...
```

4. Fixed stars functions

4.1 swe_fixstar_ut

The function swe_fixstar_ut() was introduced with Swisseph **version 1.60**. It does exactly the same as swe_fixstar() except that it expects Universal Time rather than Ephemeris time as an input value. (cf. swe_calc_ut() and swe_calc())
The functions swe_fixstar_ut() and swe_fixstar() computes fixed stars. They are defined as follows:

```
long swe_fixstar_ut(char* star, double tjd_ut, long iflag, double* xx, char* serr);
```

where
star =name of fixed star to be searched, returned name of found star
tjd_ut =Julian day in Universal Time
iflag =an integer containing several flags that indicate what kind of computation is wanted
xx =array of 6 doubles for longitude, latitude, distance, speed in long., speed in lat., and speed in dist.
serr[256] =character string to contain error messages in case of error.
For more info, see below under 4.2. swe_fixstar()

4.2 swe_fixstar()

```
long swe_fixstar(char *star, double tjd_et, long iflag, double* xx, char* serr);
same, but  tjd_et= Julian day in Ephemeris Time
```

The parameter **star** must provide for at least 41 characters for the returned star name (= 2 x SE_MAX_STNAME + 1, where SE_MAX_STNAME is defined in swephexp.h). If a star is found, its name is returned in this field in the format
traditional_name, nomenclature_name e.g. "Aldebaran,alTau".

The function has three modes to search for a star in the file fixstars.cat:

- **star** contains a positive number (in ASCII string format, e.g. "234"): The 234-th non-comment line in the file fixstars.cat is used. Comment lines begin with # and are ignored.
- **star** contains a traditional name: the first star in the file fixstars.cat is used whose traditional name fits the given name. All names are mapped to lower case before comparison. If **star** has **n** characters, only the first **n** characters of the traditional name field are compared. If a comma appears after a non-zero-length traditional name, the traditional name is cut off at the comma before the search. This allows the reuse of the returned star name from a previous call in the next call.
- **star** begins with a comma, followed by a nomenclature name, e.g. ",alTau": the star with this name in the nomenclature field (the second field) is returned. Letter case is observed in the comparison for nomenclature names.

For correct spelling of nomenclature names, see file fixstars.cat. Nomenclature names are usually composed of a Greek letter and the name of a star constellation. The Greek letters were originally used to write numbers, therefore to number the stars of the constellation. The abbreviated nomenclature names we use in fixstars.cat are constructed from two lowercase letters for the Greek letter (e.g. ″al″ for ″alpha″) and three letters for the constellation (e.g. ″Tau″ for ″Tauri″).

The function and the DLL should survive damaged fixstars.cat files which contain illegal data and star names exceeding the accepted length. Such fields are cut to acceptable length.
There are two special entries in the file fixstars.cat:

- an entry for the Galactic Center, named "Gal. Center" with one blank.

- a star named "AA_page_B40" which is the star calculation sample of Astronomical Almanac (our bible of the last two years), page B40.

You may edit the star catalogue and move the stars you prefer to the top of the file. This will increase the speed of your computations. The search mode is linear through the whole star file for each call of swe_fixstar().
As for the explanation of the other parameters, see swe_calc().
Barycentric positions are not implemented. The difference between geocentric and heliocentric fix star position is noticeable and arises from parallax and gravitational deflection.
Attention: swe_fixstar() **does not compute speeds** of the fixed stars. If you need them, you have to compute them on your own, calling swe_fixstar() for a second (and third) time.

4.3 swe_fixstar_mag()

```
long swe_fixstar_mag(char *star, double* mag, char* serr);
```

Function calculates the magnitude of a fixed star. The function returns OK or ERR. The magnitude value is returned in the parameter mag.
For the definition and use of the parameter star see function swe_fixstar(). The parameter serr and is, as usually, an error string pointer.

5. Apsides functions

5.1 swe_nod_aps_ut

The functions swe_nod_aps_ut() and swe_nod_aps() compute planetary nodes and apsides (perihelia, aphelia, second focal points of the orbital ellipses). Both functions do exactly the same except that they expect a different time parameter (cf. swe_calc_ut() and swe_calc()).

The definitions are:

```
int32 swe_nod_aps_ut(double tjd_ut, int32 ipl, int32 iflag, int32 method, double *xnasc, double
      *xndsc, double *xperi, double *xaphe, char *serr);
```

where

```
tjd_ut      =Julian day in Universal Time
ipl         =planet number
iflag       =same as with swe_calc_ut() and swe_fixstar_ut()
method      =another integer that specifies the calculation method, see explanations below
xnasc       =array of 6 doubles for ascending node
xndsc       =array of 6 doubles for descending node
xperi       =array of 6 doubles for perihelion
xaphe       =array of 6 doubles for aphelion
serr[256]   =character string to contain error messages in case of error.
```

5.2 swe_nod_aps()

```
int32 swe_nod_aps(double tjd_et, int32 ipl, int32 iflag, int32 method, double *xnasc, double *xndsc,
      double *xperi, double *xaphe, char *serr);
```

same, but

```
tjd_et =          Julian day in Ephemeris Time
```

The parameter **iflag** allows the same specifications as with the function swe_calc_ut(). I.e., it contains the Ephemeris flag, the heliocentric, topocentric, speed, nutation flags etc. etc.
The parameter **method** tells the function what kind of nodes or apsides are required:

```
#define SE_NODBIT_MEAN          1
```

This is also the default. Mean nodes and apsides are calculated for the bodies that have them, i.e. for the Moon and the planets Mercury through Neptune, osculating ones for Pluto and the asteroids.

```
#define SE_NODBIT_OSCU          2
```

Osculating nodes and apsides are calculated for all bodies.

```
#define SE_NODBIT_OSCU_BAR      4
```

Osculating nodes and apsides are calculated for all bodies. With planets beyond Jupiter, they are computed from a barycentric ellipse. Cf. the explanations in swisseph.doc.

If this bit is combined with SE_NODBIT_MEAN, mean values are given for the planets Mercury - Neptun.

```
#define SE_NODBIT_FOPOINT   256
```

The second focal point of the orbital ellipse is computed and returned in the array of the aphelion. This bit can be combined with any other bit.

It is not meaningful to compute mean oribital elements topocentrically. The concept of mean elements precludes consideration of any short term fluctuations in coordinates.

6. Eclipse and planetary phenomena functions

There are the following functions for eclipse and occultation calculations.

Solar eclipses:
- swe_sol_eclipse_when_loc(tjd...) finds the next eclipse for a given geographic position.
- swe_sol_eclipse_when_glob(tjd...) finds the next eclipse globally.
- swe_sol_eclipse_where() computes the geographic location of a solar eclipse for a given tjd.
- swe_sol_eclipse_how() computes attributes of a solar eclipse for a given tjd, geographic longitude, latitude and height.

Occultations of planets by the moon:
These functions can also be used for solar eclipses. But they are slightly less efficient.
- swe_lun_occult_when_loc(tjd...) finds the next occultation for a body and a given geographic position.
- swe_lun_occult_when_glob(tjd...) finds the next occultation of a given body globally.
- swe_lun_occult_where() computes the geographic location of an occultation for a given tjd.

Lunar eclipses:
- swe_lun_eclipse_when(tjd...) finds the next lunar eclipse.
- swe_lun_eclipse_how() computes the attributes of a lunar eclipse for a given tjd.

Risings, settings, and meridian transits of planets and stars:
- swe_rise_trans()
- swe_rise_trans_true_hor( ) returns rising and setting times for a local horizon with altitude != 0

Planetary phenomena:
- swe_pheno_ut() and swe_pheno() compute phase angle, phase, elongation, apparent diameter, and apparent magnitude of the Sun, the Moon, all planets and asteroids.

6.0. Example of a typical eclipse calculation

Find the next total eclipse, calculate the geographical position where it is maximal and the four contacts for that position (for a detailed explanation of all eclipse functions see the next chapters):

```
double tret[10], attr[20], geopos[10];
char serr[255];
int32 whicheph = 0; /* default ephemeris */
double tjd_start = 2451545; /* Julian day number for 1 Jan 2000 */
int32 ifltype = SE_ECL_TOTAL ¦ SE_ECL_CENTRAL ¦ SE_ECL_NONCENTRAL;
/* find next eclipse anywhere on earth */
eclflag = swe_sol_eclipse_when_glob(tjd_start, whicheph,  ifltype, tret, 0, serr);
if (eclflag == ERR)
  return ERR;
/* the time of the greatest eclipse has been returned in tret[0];
 * now we can find geographical position of the eclipse maximum */
tjd_start = tret[0];
eclflag = swe_sol_eclipse_where(tjd_start, whicheph, geopos, attr, serr);
if (eclflag == ERR)
  return ERR;
/* the geographical position of the eclipse maximum is in geopos[0] and geopos[1];
 * now we can calculate the four contacts for this place. The start time is chosen
 * a day before the maximum eclipse: */
tjd_start = tret[0] - 1;
eclflag = swe_sol_eclipse_when_loc(tjd_start, whicheph, geopos, tret, attr, 0, serr);
if (eclflag == ERR)
  return ERR;
/* now tret[] contains the following values:
 * tret[0] = time of greatest eclipse (Julian day number)
 * tret[1] = first contact
 * tret[2] = second contact
 * tret[3] = third contact
```

```
 * tret[4] = fourth contact */
```

6.1. swe_sol_eclipse_when_loc() and swe_lun_occult_when_loc()

To find the next eclipse for a given geographic position, use swe_sol_eclipse_when_loc().

```
int32 swe_sol_eclipse_when_loc(
double tjd_start,      /* start date for search, Jul. day UT */
int32 ifl,             /* ephemeris flag */
double *geopos,        /* 3 doubles for geo. lon, lat, height eastern longitude is positive,
                          western longitude is negative,  northern latitude is positive,
                          southern latitude is negative */
double *tret,          /* return array, 10 doubles, see below */
double *attr,          /* return array, 20 doubles, see below */
AS_BOOL backward,      /* TRUE, if backward search */
char *serr);           /* return error string */
```

The function returns:

```
/* retflag       -1 (ERR) on error (e.g. if swe_calc() for sun or moon fails)
        SE_ECL_TOTAL or SE_ECL_ANNULAR or SE_ECL_PARTIAL
        SE_ECL_VISIBLE,
        SE_ECL_MAX_VISIBLE,
        SE_ECL_1ST_VISIBLE, SE_ECL_2ND_VISIBLE
        SE_ECL_3ST_VISIBLE, SE_ECL_4ND_VISIBLE

 tret[0]       time of maximum eclipse
 tret[1]       time of first contact
 tret[2]       time of second contact
 tret[3]       time of third contact
 tret[4]       time of forth contact
 tret[5]       time of sunrise between first and forth contact (not implemented so far)
 tret[6]       time of sunset beween first and forth contact  (not implemented so far)

 attr[0]        fraction of solar diameter covered by moon;
                with total/annular eclipses, it results in magnitude acc. to IMCCE.
 attr[1]        ratio of lunar diameter to solar one
 attr[2]        fraction of solar disc covered by moon (obscuration)
 attr[3]        diameter of core shadow in km
 attr[4]        azimuth of sun at tjd
 attr[5]        true altitude of sun above horizon at tjd
 attr[6]        apparent altitude of sun above horizon at tjd
 attr[7]        elongation of moon in degrees
 attr[8]         magnitude acc. to NASA;
                 = attr[0] for partial and attr[1] for annular and total eclipses
 attr[9]       saros series number
 attr[10]      saros series member number
*/
```

6.2. swe_sol_eclipse_when_glob()

To find the next eclipse globally:

```
int32 swe_sol_eclipse_when_glob(
double tjd_start,       /* start date for search, Jul. day UT */
int32 ifl,               /* ephemeris flag */
int32 ifltype,          /* eclipse type wanted: SE_ECL_TOTAL etc. or 0, if any eclipse type */
double *tret,           /* return array, 10 doubles, see below */
AS_BOOL backward,       /* TRUE, if backward search */
char *serr);             /* return error string */
```

This function requires the time parameter *tjd_start* in *Universal Time* and also yields the return values (*tret[]*) in UT. For conversions between ET and UT, use the function swe_deltat().

Note: An implementation of this function with parameters in Ephemeris Time would have been possible. The question when the next solar eclipse will happen anywhere on earth is independent of the rotational position of the earth and therefore independent of Delta T. However, the function is often used in combination with other eclipse functions (see example below), for which input and output in ET makes no sense, because they concern local circumstances of an eclipse and therefore *are* dependent on the rotational position of the earth. For this reason, UT has been chosen for the time parameters of all eclipse functions.

ifltype specifies the eclipse type wanted. It can be a combination of the following bits (see swephexp.h):

```
#define SE_ECL_CENTRAL          1
#define SE_ECL_NONCENTRAL       2
#define SE_ECL_TOTAL            4
#define SE_ECL_ANNULAR          8
#define SE_ECL_PARTIAL          16
#define SE_ECL_ANNULAR_TOTAL    32
```

Recommended values for ifltype:

```
/* search for any eclipse, no matter which type */
ifltype = 0;
/* search a total eclipse; note: non-central total eclipses are very rare */
ifltype = SE_ECL_TOTAL ¦ SE_ECL_CENTRAL ¦ SE_ECL_NONCENTRAL;
/* search an annular eclipse */
ifltype = SE_ECL_TOTAL ¦ SE_ECL_CENTRAL ¦ SE_ECL_NONCENTRAL;
/* search an annular-total (hybrid) eclipse */
ifltype_ = SE_ECL_ANNULAR_TOTAL ¦ SE_ECL_CENTRAL ¦ SE_ECL_NONCENTRAL;
/* search a partial eclipse */
ifltype = SE_ECL_PARTIAL;
```

If your code does not work, please study the sample code in swetest.c.

The function returns:

```
/* retflag        -1 (ERR) on error (e.g. if swe_calc() for sun or moon fails)
       SE_ECL_TOTAL or SE_ECL_ANNULAR or SE_ECL_PARTIAL or SE_ECL_ANNULAR_TOTAL
       SE_ECL_CENTRAL
       SE_ECL_NONCENTRAL

 tret[0]        time of maximum eclipse
 tret[1]        time, when eclipse takes place at local apparent noon
 tret[2]        time of eclipse begin
 tret[3]        time of eclipse end
 tret[4]        time of totality begin
 tret[5]        time of totality end
 tret[6]        time of center line begin
 tret[7]        time of center line end
 tret[8]        time when annular-total eclipse becomes total not implemented so far
 tret[9]        time when annular-total eclipse becomes annular again not implemented so far

      declare as tret[10] at least !
 */
```

6.3. swe_sol_eclipse_how ()

To calculate the attributes of an eclipse for a given geographic position and time:

```
int32 swe_sol_eclipse_how(
double tjd_ut,      /* time, Jul. day UT */
int32 ifl,          /* ephemeris flag */
double *geopos      /* geogr. longitude, latitude, height above sea
                     * eastern longitude is positive,
                     * western longitude is negative,
                     * northern latitude is positive,
                     * southern latitude is negative */
double *attr,       /* return array, 20 doubles, see below */
char *serr);        /* return error string */
```

```
/* retflag       -1 (ERR) on error (e.g. if swe_calc() for sun or moon fails)
        SE_ECL_TOTAL or SE_ECL_ANNULAR or SE_ECL_PARTIAL
                0, if no eclipse is visible at geogr. position.

 attr[0]        fraction of solar diameter covered by moon;
                with total/annular eclipses, it results in magnitude acc. to IMCCE.
 attr[1]        ratio of lunar diameter to solar one
 attr[2]        fraction of solar disc covered by moon (obscuration)
 attr[3]        diameter of core shadow in km
 attr[4]        azimuth of sun at tjd
 attr[5]        true altitude of sun above horizon at tjd
 attr[6]        apparent altitude of sun above horizon at tjd
 attr[7]        elongation of moon in degrees
 attr[8]        magnitude acc. to NASA;
                = attr[0] for partial and attr[1] for annular and total eclipses
 attr[9]       saros series number
 attr[10]      saros series member number
```

6.4. swe_sol_eclipse_where ()

This function can be used to find out the geographic position, where, for a given time, a central eclipse is central or where a non-central eclipse is maximal.
If you want to draw the eclipse path of a total or annular eclipse on a map, first compute the start and end time of the total or annular phase with swe_sol_eclipse_when_glob(), then call swe_sol_eclipse_how() for several time intervals to get geographic positions on the central path. The northern and southern limits of the umbra and penumbra are not implemented yet.

```
int32 swe_sol_eclipse_where (
double tjd_ut,          /* time, Jul. day UT */
int32 ifl,              /* ephemeris flag */
double *geopos,         /* return array, 2 doubles, geo. long. and lat.
                         * eastern longitude is positive,
                         * western longitude is negative,
                         * northern latitude is positive,
                         * southern latitude is negative */
double *attr,           /* return array, 20 doubles, see below */
char *serr);            /* return error string */
```

The function returns:

```
/* -1 (ERR)             on error (e.g. if swe_calc() for sun or moon fails)
  0     if there is no solar eclipse at tjd
  SE_ECL_TOTAL
  SE_ECL_ANNULAR
  SE_ECL_TOTAL | SE_ECL_CENTRAL
  SE_ECL_TOTAL | SE_ECL_NONCENTRAL
  SE_ECL_ANNULAR | SE_ECL_CENTRAL
  SE_ECL_ANNULAR | SE_ECL_NONCENTRAL
  SE_ECL_PARTIAL

  geopos[0]:            geographic longitude of central line
  geopos[1]:            geographic latitude of central line

  not implemented so far:
  geopos[2]:            geographic longitude of northern limit of umbra
  geopos[3]:            geographic latitude of northern limit of umbra
  geopos[4]:            geographic longitude of southern limit of umbra
  geopos[5]:            geographic latitude of southern limit of umbra
  geopos[6]:            geographic longitude of northern limit of penumbra
  geopos[7]:            geographic latitude of northern limit of penumbra
  geopos[8]:            geographic longitude of southern limit of penumbra
  geopos[9]:            geographic latitude of southern limit of penumbra

  eastern longitudes are positive,
  western longitudes are negative,
  northern latitudes are positive,
```

```
        southern latitudes are negative

        attr[0]         fraction of solar diameter covered by the moon
        attr[1]         ratio of lunar diameter to solar one
        attr[2]         fraction of solar disc covered by moon (obscuration)
        attr[3]         diameter of core shadow in km
        attr[4]         azimuth of sun at tjd
        attr[5]         true altitude of sun above horizon at tjd
        attr[6]         apparent altitude of sun above horizon at tjd
        attr[7]         angular distance of moon from sun in degrees
        attr[8]         eclipse magnitude (= attr[0] or attr[1] depending on eclipse type)
        attr[9]         saros series number
        attr[10]        saros series member number

            declare as attr[20]!
      */
```

6.5. swe_lun_occult_when_loc()

To find the next occultation of a planet or star by the moon for a given location, use swe_lun_occult_when_loc().
The same function can also be used for local solar eclipses instead of swe_sol_eclipse_when_loc(), but is a bit less efficient.

```
/* Same declaration as swe_sol_eclipse_when_loc().
 * In addition:
 * int32 ipl            planet number of occulted body
 * char* starname       name of occulted star. Must be NULL or "", if a planetary
 *                      occultation is to be calculated. For use of this field,
 *                      see swe_fixstar().
 * int32 ifl            ephemeris flag. If you want to have only one conjunction
 *                      of the moon with the body tested, add the following flag:
 *                      backward |= SE_ECL_ONE_TRY. If this flag is not set,
 *                      the function will search for an occultation until it
 *                      finds one. For bodies with ecliptical latitudes > 5,
 *                      the function may search successlessly until it reaches
 *                      the end of the ephemeris.
 */
int32 swe_lun_occult_when_loc(
double tjd_start,     /* start date for search, Jul. day UT */
int32 ipl,            /* planet number */
char* starname,       /* star name, must be NULL or "" if not a star */
int32 ifl,            /* ephemeris flag */
double *geopos,       /* 3 doubles for geo. lon, lat, height eastern longitude is positive,
                         western longitude is negative,  northern latitude is positive,
                         southern latitude is negative */
double *tret,         /* return array, 10 doubles, see below */
double *attr,         /* return array, 20 doubles, see below */
AS_BOOL backward,     /* TRUE, if backward search */
char *serr);          /* return error string */
```

If an occultation of *any* planet is wanted, call the function for all planets you want to consider and find the one with the smallest tret[1] (first contact). (If searching backward, find the one with the greatest tret[1]). For efficiency, set ifl |= SE_ECL_ONE_TRY. With this flag, only the next conjunction of the moon with the bodies is checked. If no occultation has been found, repeat the calculation with tstart = tstart + 20.

The function returns:

```
/* retflag
        -1 (ERR) on error (e.g. if swe_calc() for sun or moon fails)
        0  (if no occultation/no eclipse found)
        SE_ECL_TOTAL or SE_ECL_ANNULAR or SE_ECL_PARTIAL
        SE_ECL_VISIBLE,
        SE_ECL_MAX_VISIBLE,
        SE_ECL_1ST_VISIBLE, SE_ECL_2ND_VISIBLE
        SE_ECL_3ST_VISIBLE, SE_ECL_4ND_VISIBLE
  These return values (except the SE_ECL_ANNULAR) also appear with occultations.
```

```
tret[0]      time of maximum eclipse
tret[1]      time of first contact
tret[2]      time of second contact
tret[3]      time of third contact
tret[4]      time of forth contact
tret[5]      time of sunrise between first and forth contact (not implemented so far)
tret[6]      time of sunset beween first and forth contact  (not implemented so far)

attr[0]      fraction of solar diameter covered by moon (magnitude)
attr[1]      ratio of lunar diameter to solar one
attr[2]      fraction of solar disc covered by moon (obscuration)
attr[3]      diameter of core shadow in km
attr[4]      azimuth of sun at tjd
attr[5]      true altitude of sun above horizon at tjd
attr[6]      apparent altitude of sun above horizon at tjd
attr[7]      elongation of moon in degrees     */
```

6.6. swe_lun_occult_when_glob()

To find the next occultation of a planet or star by the moon globally (not for a particular geographic location), use swe_lun_occult_when_glob().
The same function can also be used for global solar eclipses instead of swe_sol_eclipse_when_glob(), but is a bit less efficient.

```
/* Same declaration as swe_sol_eclipse_when_glob().
 * In addition:
 * int32 ipl              planet number of occulted body
 * char* starname         name of occulted star. Must be NULL or "", if a planetary
 *                        occultation is to be calculated. For use of this field,
 *                        see swe_fixstar().
 * int32 ifl              ephemeris flag. If you want to have only one conjunction
 *                        of the moon with the body tested, add the following flag:
 *                        backward |= SE_ECL_ONE_TRY. If this flag is not set,
 *                        the function will search for an occultation until it
 *                        finds one. For bodies with ecliptical latitudes > 5,
 *                        the function may search successlessly until it reaches
 *                        the end of the ephemeris.
 */
int32 swe_lun_occult_when_glob(
double tjd_start,     /* start date for search, Jul. day UT */
int32 ipl,            /* planet number */
char* starname,       /* star name, must be NULL or "" if not a star */
int32 ifl,            /* ephemeris flag */
int32 ifltype,        /* eclipse type wanted */
double *geopos,       /* 3 doubles for geo. lon, lat, height eastern longitude is positive,
                         western longitude is negative,  northern latitude is positive,
                         southern latitude is negative */
double *tret,         /* return array, 10 doubles, see below */
AS_BOOL backward,     /* TRUE, if backward search */
char *serr);          /* return error string */
```

If an occultation of *any* planet is wanted, call the function for all planets you want to consider and find the one with the smallest tret[1] (first contact). (If searching backward, find the one with the greatest tret[1]). For efficiency, set ifl |= SE_ECL_ONE_TRY. With this flag, only the next conjunction of the moon with the bodies is checked. If no occultation has been found, repeat the calculation with tstart = tstart + 20.

The function returns:

```
/* retflag
       -1 (ERR) on error (e.g. if swe_calc() for sun or moon fails)
       0  (if no occultation / eclipse has been found)
       SE_ECL_TOTAL or SE_ECL_ANNULAR or SE_ECL_PARTIAL or SE_ECL_ANNULAR_TOTAL
       SE_ECL_CENTRAL
       SE_ECL_NONCENTRAL
```

```
      tret[0]      time of maximum eclipse
      tret[1]      time, when eclipse takes place at local apparent noon
      tret[2]      time of eclipse begin
      tret[3]      time of eclipse end
      tret[4]      time of totality begin
      tret[5]      time of totality end
      tret[6]      time of center line begin
      tret[7]      time of center line end
      tret[8]      time when annular-total eclipse becomes total not implemented so far
      tret[9]      time when annular-total eclipse becomes annular again not implemented so far

          declare as tret[10] at least !
    */
```

6.7. swe_lun_occult_where ()

Similar to swe_sol_eclipse_where(), this function can be used to find out the geographic position, where, for a given time, a central eclipse is central or where a non-central eclipse is maximal. With occultations, it tells us, at which geographic location the occulted body is in the middle of the lunar disc or closest to it. Because occultations are always visible from a very large area, this is not very interesting information. But it may become more interesting as soon as the limits of the umbra (and penumbra) will be implemented.

```
    int32 swe_lun_occult_where (
    double tjd_ut,          /* time, Jul. day UT */
    int32 ipl,          /* planet number */
    char* starname,     /* star name, must be NULL or "" if not a star */
    int32 ifl,          /* ephemeris flag */
    double *geopos,     /* return array, 2 doubles, geo. long. and lat.
                         * eastern longitude is positive,
                         * western longitude is negative,
                         * northern latitude is positive,
                         * southern latitude is negative */
    double *attr,       /* return array, 20 doubles, see below */
    char *serr);        /* return error string */
```

The function returns:

```
    /* -1 (ERR)          on error (e.g. if swe_calc() for sun or moon fails)
      0     if there is no solar eclipse (occultation) at tjd
      SE_ECL_TOTAL
      SE_ECL_ANNULAR
      SE_ECL_TOTAL | SE_ECL_CENTRAL
      SE_ECL_TOTAL | SE_ECL_NONCENTRAL
      SE_ECL_ANNULAR | SE_ECL_CENTRAL
      SE_ECL_ANNULAR | SE_ECL_NONCENTRAL
      SE_ECL_PARTIAL

      geopos[0]:          geographic longitude of central line
      geopos[1]:          geographic latitude of central line

      not implemented so far:
      geopos[2]:          geographic longitude of northern limit of umbra
      geopos[3]:          geographic latitude of northern limit of umbra
      geopos[4]:          geographic longitude of southern limit of umbra
      geopos[5]:          geographic latitude of southern limit of umbra
      geopos[6]:          geographic longitude of northern limit of penumbra
      geopos[7]:          geographic latitude of northern limit of penumbra
      geopos[8]:          geographic longitude of southern limit of penumbra
      geopos[9]:          geographic latitude of southern limit of penumbra

      eastern longitudes are positive,
      western longitudes are negative,
      northern latitudes are positive,
      southern latitudes are negative

      attr[0]             fraction of solar diameter covered by moon (magnitude)
```

```
      attr[1]          ratio of lunar diameter to solar one
      attr[2]          fraction of solar disc covered by moon (obscuration)
      attr[3]          diameter of core shadow in km
      attr[4]          azimuth of sun at tjd
      attr[5]          true altitude of sun above horizon at tjd
      attr[6]          apparent altitude of sun above horizon at tjd
      attr[7]          angular distance of moon from sun in degrees

          declare as attr[20]!
    */
```

6.8. swe_lun_eclipse_when ()

To find the next lunar eclipse:

```
int32 swe_lun_eclipse_when(
double tjd_start,       /* start date for search, Jul. day UT */
int32 ifl,              /* ephemeris flag */
int32 ifltype,          /* eclipse type wanted: SE_ECL_TOTAL etc.  or 0, if any eclipse type */
double *tret,           /* return array, 10 doubles, see below */
AS_BOOL backward,       /* TRUE, if backward search */
char *serr);            /* return error string */
```

Recommended values for ifltype:

```
/* search for any lunar eclipse, no matter which type */
ifltype = 0;
/* search a total lunar eclipse */
ifltype = SE_ECL_TOTAL;
/* search a partial lunar eclipse */
ifltype = SE_ECL_PARTIAL;
/* search a penumbral lunar eclipse */
ifltype = SE_ECL_PENUMBRAL;
```

If your code does not work, please study the sample code in swetest.c.

The function returns:

```
/* retflag              -1 (ERR) on error (e.g. if swe_calc() for sun or moon fails)
       SE_ECL_TOTAL or SE_ECL_PENUMBRAL or SE_ECL_PARTIAL
  tret[0]               time of maximum eclipse
  tret[1]
  tret[2]               time of partial phase begin (indices consistent with solar eclipses)
  tret[3]               time of partial phase end
  tret[4]               time of totality begin
  tret[5]               time of totality end
  tret[6]               time of penumbral phase begin
  tret[7]               time of penumbral phase end
 */
```

6.9. swe_lun_eclipse_how ()

This function computes the attributes of a lunar eclipse at a given time:

```
int32 swe_lun_eclipse_how(
double tjd_ut,          /* time, Jul. day UT */
int32 ifl,              /* ephemeris flag */
double *geopos,         /* input array, geopos, geolon, geoheight
                           eastern longitude is positive,
                           western longitude is negative,
                           northern latitude is positive,
                           southern latitude is negative */
double *attr,           /* return array, 20 doubles, see below */
char *serr);            /* return error string */
```

The function returns:

```
/* retflag              -1 (ERR) on error (e.g. if swe_calc() for sun or moon fails)
        SE_ECL_TOTAL or SE_ECL_PENUMBRAL or SE_ECL_PARTIAL
                        0       if there is no eclipse

attr[0]                 umbral magnitude at tjd
attr[1]                 penumbral magnitude
attr[4]                 azimuth of moon at tjd. Not implemented so far
attr[5]                 true altitude of moon above horizon at tjd. Not implemented so far
attr[6]                 apparent altitude of moon above horizon at tjd. Not implemented so far
attr[7]                 distance of moon from opposition in degrees
attr[8]                 eclipse magnitude (= attr[0])
attr[9]                 saros series number
attr[10]                saros series member number

        declare as attr[20] at least !
 */
```

6.10. swe_rise_trans() and swe_rise_trans_true_hor() (risings, settings, meridian transits)

The function swe_rise_trans() computes the times of rising, setting and meridian transits for all planets, asteroids, the moon, and the fixed stars. The function swe_rise_trans_true_hor() does the same for a local horizon that has an altitude != 0. Their definitions are as follows:

```
int32 swe_rise_trans(
double tjd_ut,          /* search after this time (UT) */
int32 ipl,              /* planet number, if planet or moon */
char *starname,         /* star name, if star */
int32 epheflag,         /* ephemeris flag */
int32 rsmi,             /* integer specifying that rise, set, orone of the two meridian transits is
                           wanted. see definition below */
double *geopos,         /* array of three doubles containing
                           * geograph. long., lat., height of observer */
double atpress,         /* atmospheric pressure in mbar/hPa */
double attemp,          /* atmospheric temperature in deg. C */
double *tret,           /* return address (double) for rise time etc. */
char *serr);            /* return address for error message */

int32 swe_rise_trans_true_hor(
double tjd_ut,          /* search after this time (UT) */
int32 ipl,              /* planet number, if planet or moon */
char *starname,         /* star name, if star */
int32 epheflag,         /* ephemeris flag */
int32 rsmi,             /* integer specifying that rise, set, orone of the two meridian transits is
                           wanted. see definition below */
double *geopos,         /* array of three doubles containing
                           * geograph. long., lat., height of observer */
double atpress,         /* atmospheric pressure in mbar/hPa */
double attemp,          /* atmospheric temperature in deg. C */
double horhgt,          /* height of local horizon in deg at the point where the body rises or sets*/
double *tret,           /* return address (double) for rise time etc. */
char *serr);            /* return address for error message */
```

The second function has one additional parameter horhgt for the height of the local horizon at the point where the body rises or sets.

The variable **rsmi** can have the following values:

```
/* for swe_rise_transit() and swe_rise_transit_true_hor() */
#define SE_CALC_RISE        1
#define SE_CALC_SET         2
#define SE_CALC_MTRANSIT    4     /* upper meridian transit (southern for northern geo. latitudes) */
#define SE_CALC_ITRANSIT    8     /* lower meridian transit (northern, below the horizon) */
/* the following bits can be added (or'ed) to SE_CALC_RISE or SE_CALC_SET */
```

```
#define SE_BIT_DISC_CENTER       256    /* for rising or setting of disc center */
#define SE_BIT_DISC_BOTTOM      8192    /* for rising or setting of lower limb of disc */
#define SE_BIT_NO_REFRACTION     512    /* if refraction is not to be considered */
#define SE_BIT_CIVIL_TWILIGHT   1024    /* in order to calculate civil twilight */
#define SE_BIT_NAUTIC_TWILIGHT 2048    /* in order to calculate nautical twilight */
#define SE_BIT_ASTRO_TWILIGHT   4096    /* in order to calculate astronomical twilight */
#define SE_BIT_FIXED_DISC_SIZE (16*1024) /* neglect the effect of distance on disc size */
```

rsmi = 0 will return risings.
The rising times depend on the atmospheric pressure and temperature. **atpress** expects the atmospheric pressure in millibar (hectopascal); **attemp** the temperature in degrees Celsius.
If **atpress** is given the value 0, the function estimates the pressure from the geographical altitude given in **geopos[2]** and **attemp**. If **geopos[2]** is 0, **atpress** will be estimated for sea level.

6.11. swe_pheno_ut() and swe_pheno(), planetary phenomena

These functions compute phase, phase angle, elongation, apparent diameter, apparent magnitude for the Sun, the Moon, all planets and asteroids. The two functions do exactly the same but expect a different time parameter.

```
int32 swe_pheno_ut(
double tjd_ut,          /* time Jul. Day UT */
int32 ipl,              /* planet number */
int32 iflag,            /* ephemeris flag */
double *attr,           /* return array, 20 doubles, see below */
char *serr);            /* return error string */

int32 swe_pheno(
double tjd_et,          /* time Jul. Day ET */
int32 ipl,              /* planet number */
int32 iflag,            /* ephemeris flag */
double *attr,           /* return array, 20 doubles, see below */
char *serr);            /* return error string */
```

The function returns:

```
/*
  attr[0] = phase angle (earth-planet-sun)
  attr[1] = phase (illumined fraction of disc)
  attr[2] = elongation of planet
  attr[3] = apparent diameter of disc
  attr[4] = apparent magnitude

        declare as attr[20] at least !

  Note: the lunar magnitude is quite a complicated thing,
  but our algorithm is very simple.
  The phase of the moon, its distance from the earth and
  the sun is considered, but no other factors.

  iflag also allows SEFLG_TRUEPOS, SEFLG_HELCTR
 */
```

6.12. swe_azalt(), horizontal coordinates, azimuth, altitude

swe_azalt() computes the horizontal coordinates (azimuth and altitude) of a planet or a star from either ecliptical or equatorial coordinates.

```
void swe_azalt(
    double tjd_ut,     // UT
    int32 calc_flag,   // SE_ECL2HOR or SE_EQU2HOR
    double *geopos,  // array of 3 doubles: geograph. long., lat., height
    double atpress,   // atmospheric pressure in mbar (hPa)
```

```
        double attemp,   // atmospheric temperature in degrees Celsius
        double *xin,     // array of 3 doubles: position of body in either  ecliptical or equatorial coordinates,
                         // depending on calc_flag
        double *xaz);    // return array of 3 doubles, containing azimuth, true altitude, apparent altitude
```

If **calc_flag**=SE_ECL2HOR, set xin[0]= ecl. long., xin[1]= ecl. lat., (xin[2]=distance (not required));
else
if **calc_flag**= SE_EQU2HOR, set xin[0]=rectascension, xin[1]=declination, (xin[2]= distance (not required));

```
        #define SE_ECL2HOR    0
        #define SE_EQU2HOR    1
```

The return values are:

```
        xaz[0] = azimuth, i.e. position degree, measured from the south point to west.
        xaz[1] = true altitude above horizon in degrees.
        xaz[2] = apparent (refracted) altitude above horizon in degrees.
```

The apparent altitude of a body depends on the atmospheric pressure and temperature. If only the true altitude is required, these parameters can be neglected.
If **atpress** is given the value 0, the function estimates the pressure from the geographical altitude given in geopos[2] and **attemp**. If geopos[2] is 0, **atpress** will be estimated for sea level.

6.13. swe_azalt_rev()

The function swe_azalt_rev() is not precisely the reverse of swe_azalt(). It computes either ecliptical or equatorial coordinates from azimuth and true altitude. If only an apparent altitude is given, the true altitude has to be computed first with the function swe_refrac() (see below).
It is defined as follows:

```
        void swe_azalt_rev(
            double tjd_ut,
            int32 calc_flag,      /* either SE_HOR2ECL or SE_HOR2EQU */
            double *geopos,       /* array of 3 doubles for geograph. pos. of observer */
            double *xin,          /* array of 2 doubles for azimuth and true altitude of planet */
            double *xout);        // return array of 2 doubles for either ecliptic or
                                  // equatorial coordinates, depending on calc_flag
```

For the definition of the azimuth and true altitude, see chapter 4.9 on swe_azalt().

```
        #define SE_HOR2ECL    0
        #define SE_HOR2EQU    1
```

6.14. swe_refrac(), swe_refract_extended(), refraction

The refraction function swe_refrac() calculates either the true altitude from the apparent altitude or the apparent altitude from the apparent altitude. Its definition is:

```
        double swe_refrac(
        double inalt,
        double atpress,          /* atmospheric pressure in mbar (hPa) */
        double attemp,           /* atmospheric temperature in degrees Celsius */
        int32 calc_flag);        /* either SE_TRUE_TO_APP or SE_APP_TO_TRUE */
```

where

```
        #define SE_TRUE_TO_APP        0
        #define SE_APP_TO_TRUE        1
```

The refraction depends on the atmospheric pressure and temperature at the location of the observer.
If **atpress** is given the value 0, the function estimates the pressure from the geographical altitude given in geopos[2] and **attemp.** If geopos[2] is 0, **atpress** will be estimated for sea level.

There is also a more sophisticated function swe_refrac_extended(), It allows correct calculation of refraction for altitudes above sea > 0, where the ideal horizon and planets that are visible may have a negative height. (for swe_refrac(), negative apparent heights do not exist!)

```
        double swe_refract_extended(
        double inalt,            /* altitude of object above geometric horizon in degrees, where
```

```
                                geometric horizon = plane perpendicular to gravity */
        double geoalt,          /* altitude of observer above sea level in meters */
        double atpress,         /* atmospheric pressure in mbar (hPa) */
        double lapse_rate,      /* (dattemp/dgeoalt) = [°K/m] */
        double attemp,          /* atmospheric temperature in degrees Celsius */
        int32 calc_flag);       /* either SE_TRUE_TO_APP or SE_APP_TO_TRUE */
```

function returns:
case 1, conversion from true altitude to apparent altitude:
- apparent altitude, if body appears above is observable above ideal horizon
- true altitude (the input value), otherwise
  "ideal horizon" is the horizon as seen above an ideal sphere (as seen from a plane over the ocean with a clear sky)

case 2, conversion from apparent altitude to true altitude:
- the true altitude resulting from the input apparent altitude, if this value is a plausible apparent altitude, i.e. if it is a position above the ideal horizon
- the input altitude otherwise

in addition the array dret[] returns the following values
- dret[0] true altitude, if possible; otherwise input value
- dret[1] apparent altitude, if possible; otherwise input value
- dret[2] refraction
- dret[3] dip of the horizon

The body is above the horizon if the dret[0] != dret[1]

6.15. Heliacal risings etc.: swe_heliacal_ut()

The function swe_heliacal_ut() the Julian day of the next heliacal phenomenon after a given start date. It works between geographic latitudes 60s – 60n.

```
        int32 swe_heliacal_ut(
        double tjdstart,        /* Julian day number of start date for the search of the heliacal event */
        double *dgeo            /* geographic position (details below) */
        double *datm,           /* atmospheric conditions (details below) */
        double *dobs,           /* observer description (details below) */
        char *objectname,       /* name string of fixed star or planet */
        int32 event_type,       /* event type (details below) */
        int32 helflag,          /* calculation flag, bitmap (details below) */
        double *dret,           /* result: array of at least 50 doubles, of which 3 are used at the moment */
        char * serr             /* error string */
        );
```

Function returns OK or ERR

Details for dgeo[] (array of doubles):

```
dgeo[0]: geographic longitude
dgeo[1]: geographic latitude
dgeo[2]: geographic altitude (eye height) in meters
```

Details for datm[] (array of doubles):

```
datm[0]: atmospheric pressure in mbar (hPa)
datm[1]: atmospheric temperature in degrees Celsius
datm[2]: relative humidity in %
datm[3]: if datm[3]>=1, then it is Meteorological Range [km]
         if 1>datm[3]>0, then it is the total atmsopheric coeffcient (ktot)
         datm[3]=0, then the other atmospheric parameters determine the total
                                                    atmsopheric coeffcient (ktot)
```

Default values:
If this is too much for you, set all these values to 0. The software will then set the following defaults: Pressure 1013.25, temperature 15, relative humidity 40. The values will be modified depending on the altitude of the observer above sea level.
If the extinction coefficient (meteorological range) datm[3] is 0, the software will calculate its value from datm[0..2].

Details for dobs[] (array of doubles):

dobs[0]: age of observer in years (default = 36)
dobs[1]: Snellen ratio of observers eyes (default = 1 = normal)
The following parameters are only relevant if the flag SE_HELFLAG_OPTICAL_PARAMS is set:
dobs[2]: 0 = monocular, 1 = binocular (actually a boolean)
dobs[3]: telescope magnification: 0 = default to naked eye (binocular), 1 = naked eye
dobs[4]: optical aperture (telescope diameter) in mm
dobs[5]: optical transmission

Details for event_type:
event_type = SE_HELIACAL_RISING (1): morning first (exists for all visible planets and stars)
event_type = SE_HELIACAL_SETTING (2): evening last (exists for all visible planets and stars)
event_type = SE_EVENING_FIRST (3): evening first (exists for Mercury, Venus, and the Moon)
event_type = SE_MORNING_LAST (4): morning last (exists for Mercury, Venus, and the Moon)

Details for helflag:
helflag contains ephemeris flag, like iflag in swe_calc() etc. In addition it can contain the following bits:
SE_HELFLAG_LONG_SEARCH (128): A heliacal event is searched until found.
If this bit is NOT set and no event is found within 5 synodic periods, the function stops searching and returns ERR.
SE_HELFLAG_HIGH_PRECISION (256): More rigorous but also slower algorithms are used
SE_HELFLAG_OPTICAL_PARAMS (512): Use this with calculations for optical instruments.
Unless this bit is set, the values of dobs[2-5] are ignored.
SE_HELFLAG_NO_DETAILS (1024): provide the date, but not details like visibility start, optimum, and end.
This bit makes the program a bit faster.

Details for return array dret[] (array of doubles):
dret[0]: start visibility (Julian day number)
dret[1]: optimum visibility (Julian day number)
dret[2]: end of visibility (Julian day number)

6.16. Magnitude limit for visibility: swe_vis_limit_mag()

The function swe_vis_lim_mag() determines the limiting visual magnitude in dark skies:

```
double swe_vis_limit_mag(
double tjdut,           /* Julian day number */
double *dgeo            /* geographic position (details under swe_heliacal_ut() */
double *datm,           /* atmospheric conditions (details under swe_heliacal_ut()) */
double *dobs,           /* observer description (details under swe_heliacal_ut()) */
char *objectname,       /* name string of fixed star or planet */
int32 helflag,          /* calculation flag, bitmap (details under swe_heliacal_ut()) */
double *dret,           /* result: magnitude required of the object to be visible */
char * serr             /* error string */
);
```

Function returns
-1 on error
-2 object is below horizon
0 OK, photopic vision
&1 OK, scotopic vision
&2 OK, near limit photopic/scotopic vision

7. Date and time conversion functions

7.1 Calendar Date and Julian Day: swe_julday(), swe_date_conversion(), /swe_revjul()

These functions are needed to convert calendar dates to the astronomical time scale which measures time in Julian days.

```
double swe_julday(int year, int month, int day, double hour, int gregflag);

int swe_date_conversion (
      int y , int m , int d ,        /* year, month, day */
```

```
        double hour,            /* hours (decimal, with fraction) */
        char c,                 /* calendar 'g'[regorian]|'j'[ulian] */
        double *tjd);           /* return value for Julian day */

    void swe_revjul (
        double tjd,         /* Julian day number */
        int gregflag,       /* Gregorian calendar: 1, Julian calendar: 0 */
        int *year,          /* target addresses for year, etc. */
        int *month, int *day, double *hour);
```

swe_julday() and swe_date_conversion() compute a Julian day number from year, month, day, and hour. swe_date_conversion() checks in addition whether the date is legal. It returns OK or ERR.
swe_revjul() is the reverse function of swe_julday(). It computes year, month, day and hour from a Julian day number.

The variable **gregflag** tells the function whether the input date is Julian calendar (**gregflag** = SE_JUL_CAL) or Gregorian calendar (**gregflag** = SE_GREG_CAL).
Usually, you will set **gregflag** = SE_GREG_CAL.
The Julian day number has nothing to do with Julius Cesar, who introduced the Julian calendar, but was invented by the monk Julianus. The Julian day number tells for a given date the number of days that have passed since the creation of the world which was then considered to have happened on 1 Jan –4712 at noon. E.g. the 1.1.1900 corresponds to the Julian day number 2415020.5.
Midnight has always a JD with fraction 0.5, because traditionally the astronomical day started at noon. This was practical because then there was no change of date during a night at the telescope. From this comes also the fact that noon ephemerides were printed before midnight ephemerides were introduced early in the 20th century.

7.2. UTC and Julian day: swe_utc_time_zone(), swe_utc_to_jd(), swe_jdet_to_utc(), swe_jdut1_to_utc()

The following functions, which were introduced with Swiss Ephemeris version 1.76, do a similar job as the functions described under 7.1. The difference is that input and output times are Coordinated Universal Time (UTC). For transformations between wall clock (or arm wrist) time and Julian Day numbers, these functions are more correct. The difference is below 1 second, though.

Use these functions to convert
- local time to UTC and UTC to local time,
- UTC to a Julian day number, and
- a Julian day number to UTC.

Note, in case of leap seconds, the input or output time may be 60.9999 seconds. Input or output forms have to allow for this.

```
    /* transform local time to UTC or UTC to local time
     *
     * input:
     *   iyear ... dsec     date and time
     *   d_timezone         timezone offset
     * output:
     *   iyear_out ... dsec_out
     *
     * For time zones east of Greenwich, d_timezone is positive.
     * For time zones west of Greenwich, d_timezone is negative.
     *
     * For conversion from local time to utc, use +d_timezone.
     * For conversion from utc to local time, use -d_timezone.
     */
    void FAR PASCAL_CONV swe_ utc_time_zone(
          int32 iyear, int32 imonth, int32 iday,
          int32 ihour, int32 imin, double dsec,
          double d_timezone,
          int32 *iyear_out, int32 *imonth_out, int32 *iday_out,
          int32 *ihour_out, int32 *imin_out, double *dsec_out
          )
```

```
/* input: date and time (wall clock time), calendar flag.
 * output: an array of doubles with Julian Day number in ET (TT) and UT (UT1)
 *           an error message (on error)
 * The function returns OK or ERR.
 */
int32 swe_utc_to_jd (
      int32 iyear, int32 imonth, int32 iday,
      int32 ihour, int32 imin, double dsec,   /* note : second is a decimal */
      gregflag,           /* Gregorian calendar: 1, Julian calendar: 0 */
      dret                /* return array, two doubles:
                           * dret[0] = Julian day in ET (TT)
                           * dret[1] = Julian day in UT (UT1) */
      serr                /* error string */
)

/* input: Julian day number in ET (TT), calendar flag
 * output: year, month, day, hour, min, sec in UTC */
void swe_jdet_to_utc (
      double tjd_et,      /* Julian day number in ET (TT) */
      gregflag,           /* Gregorian calendar: 1, Julian calendar: 0 */
      int32 *iyear, int32 *imonth, int32 *iday,
      int32 *ihour, int32 *imin, double *dsec,   /* note : second is a decimal */
)

/* input: Julian day number in UT (UT1), calendar flag
 * output: year, month, day, hour, min, sec in UTC */
void swe_jdut1_to_utc (
      double tjd_ut,      /* Julian day number in ET (TT) */
      gregflag,           /* Gregorian calendar: 1, Julian calendar: 0 */
      int32 *iyear, int32 *imonth, int32 *iday,
      int32 *ihour, int32 *imin, double *dsec,   /* note : second is a decimal */
)
```

How do I get correct planetary positions, sidereal time, and house cusps, starting from a wall clock date and time?

```
int32 iday, imonth, iyear, ihour, imin, retval;
int32 gregflag = SE_GREG_CAL;
double d_timezone = 5.5 ; /* time zone = Indian Standard Time; note: east is positive */
double dsec, tjd_et, tjd_ut;
double dret[2];
char serr[256];
...
/* if date and time is in time zone different from UTC, the time zone offset must be subtracted
 * first in order to get UTC: */
swe_utc_time_zone(iyear, imonth, iday, ihour, imin, dsec, d_timezone,
            &iyear_utc, &imonth_utc, &iday_utc, &ihour_utc, &imin_utc, &dsec_utc)
/* calculate Julian day number in UT (UT1) and ET (TT) from UTC */
retval = swe_utc_to_jd (iyear_utc, imonth_utc, iday_utc, ihour_utc, imin_utc, dsec_utc, gregflag, dret,
       serr);
if (retval == ERR) {
  fprintf(stderr, serr);  /* error handling */
}
tjd_et = dret[0];  /* this is ET (TT) */
tjd_ut = dret[1];  /* this is UT (UT1) */
/* calculate planet with tjd_et */
swe_calc(tjd_et, ...);
/* calculate houses with tjd_ut */
swe_houses(tjd_ut, ...)
```

And how do you get the date and wall clock time from a Julian day number? Depending on whether you have tjd_et (Julian day as ET (TT)) or tjd_ut (Julian day as UT (UT1)), use one of the two functions swe_jdet_to_utc() or swe_jdut1_to_utc().

```
...
/* first, we calculate UTC from TT (ET) */
swe_jdet_to_utc(tjd_et, gregflag, &iyear_utc, &imonth_utc, &iday_utc, &ihour_utc, &imin_utc,
       &dsec_utc);
/* now, UTC to local time (note the negative sign before d_timezone): */
```

```
swe_utc_time_zone(iyear_utc, imonth_utc, iday_utc, ihour_utc, imin_utc, dsec_utc,
-d_timezone, &iyear, &imonth, &iday, &ihour, &imin, &dsec)
```

7.3. Future insertion of leap seconds and the file swe_leapsec.txt

The insertion of leap seconds is not known in advance. We will update the Swiss Ephemeris whenever the IERS announces that a leap second will be inserted. However, if the user does not want to wait for our update or does not want to download a new version of the Swiss Ephemeris, he can create a file swe_leapsec.txt in the ephemeris directory. Insert a line with the date on which a leap second has to be inserted. The file looks as follows:

```
# This file contains the dates of leap seconds to be taken into account
# by the Swiss Ephemeris.
# For each new leap second add the date of its insertion in the format
# yyyymmdd, e.g. "20081231" for 21 december 2008
20081231
```

7.4. Mean solar time versus True solar time: swe_time_equ()

Universal Time (UT or UTC) is based on Mean Solar Time, AKA Local Mean Time, which is a uniform measure of time. A day has always the same length, independent of the time of the year.
In the centuries before mechanical clocks where used, when the reckoning of time was mostly based on sun dials, the True Solar Time was used, also called Local Apparent Time.
The difference between Local Mean Time and Local Apparent Time is called the **equation of time**. This difference can become as large as 20 minutes.
If a birth time of a historical person was noted in Local Apparent Time, it must first be converted to Local Mean Time by applying the equation of time, before it can be used to compute Universal Time (for the houses) and finally Ephemeris Time (for the planets).

There is a function for computing the correction value.

```
/* equation of time function returns the difference between local apparent and local mean time.
 e = LAT – LMT.  tjd is ephemeris time */
int swe_time_equ(double tjd, double* e, char* serr);
```

If you first compute **tjd** on the basis of the registered Local Apparent Time, you convert it to Local Mean Time with:

```
tjd_mean = tjd_app + e;
```

8. Delta T-related functions

```
/* delta t from Julian day number */
double swe_deltat(double tjd);
/* get tidal acceleration used in swe_deltat() */
double swe_get_tid_acc(void);
/* set tidal acceleration to be used in swe_deltat() */
void swe_set_tid_acc(double t_acc);
```

The Julian day number, you compute from a birth date, will be Universal Time (UT, former GMT) and can be used to compute the star time and the houses. However, for the planets and the other factors, you have to convert UT to Ephemeris time (ET):

8.1 swe_deltat()

```
tjde = tjd + swe_deltat(tjd);   where tjd = Julian day in UT, tjde = in ET
```

For precision fanatics: The value of **delta t** depends on the tidal acceleration in the motion of the moon. Its default value corresponds to the state-of-the-art JPL ephemeris (e.g. DE406, s. swephexp.h). If you use another JPL ephemeris, e.g. DE200, you may wish the tidal constant of DE200. This makes a difference of 0.5 time seconds in 1900 and 4 seconds in 1800 (= 0.2" in the position of the sun). However, this effect is limited to the period 1620 - ~1997. To change the tidal acceleration, use the function

8.2 swe_set_tid_acc(), swe_get_tid_acc()

```
swe_set_tid_acc(acceleration);        // Do this before calling deltat() !
```

The values that **acceleration** can have are listed in swephexp.h. (e.g. SE_TIDAL_200, etc.)
To find out the built-in value of the tidal acceleration, you can call

```
acceleration = swe_get_tidacc();
```

8.3. Future updates of Delta T and the file swe_deltat.txt

Delta T values for future years can only be estimated. Strictly speaking, the Swiss Ephemeris has to be updated every year after the new Delta T value for the past year has been published by the IERS. We will do our best and hope to update the Swiss Ephemeris every year. However, if the user does not want to wait for our update or does not download a new version of the Swiss Ephemeris he can add new Delta T values in the file swe_deltat.txt, which has to be located in the Swiss Ephemeris ephemeris path.

```
# This file allows make new Delta T known to the Swiss Ephemeris.
# Note, these values override the values given in the internal Delta T
# table of the Swiss Ephemeris.
# Format: year and seconds (decimal)
2003 64.47
2004 65.80
2005 66.00
2006 67.00
2007 68.00
2008 68.00
2009 69.00
```

9. The function swe_set_topo() for topocentric planet positions

```
void swe_set_topo(double geolon, double geolat, double altitude);
                          /* eastern longitude is positive,  western longitude is negative,
                            northern latitude is positive,   southern latitude is negative */
```

This function must be called before topocentric planet positions for a certain birth place can be computed. It tells Swiss Ephemeris, what geographic position is to be used. Geographic longitude **geolon** and latitude **geolat** must be in degrees, the **altitude** above sea must be in meters. Neglecting the altitude can result in an error of about 2 arc seconds with the moon and at an altitude 3000 m. After calling swe_set_topo(), add SEFLG_TOPOCTR to **iflag** and call swe_calc() as with an ordinary computation. E.g.:

```
swe_set_topo(geo_lon, geo_lat, altitude_above_sea);
iflag | = SEFLG_TOPOCTR;

for (i = 0; i < NPLANETS; i++) {
  iflgret = swe_calc( tjd, ipl, iflag, xp, serr );
  printf("%f\n", xp[0]);
}
```

The parameters set by swe_set_topo() survive swe_close().

10. Sidereal mode functions

10.1. swe_set_sid_mode()

```
void swe_set_sid_mode (int32 sid_mode, double t0, double ayan_t0);
```

This function can be used to specify the mode for sidereal computations.
swe_calc() or swe_fixstar() has then to be called with the bit SEFLG_SIDEREAL.
If swe_set_sid_mode() is not called, the default **ayanamsha** (Fagan/Bradley) is used.
If a predefined mode is wanted, the variable **sid_mode** has to be set, while **t0** and **ayan_t0** are not considered, i.e. can be 0. The predefined sidereal modes are:

```
#define SE_SIDM_FAGAN_BRADLEY      0
#define SE_SIDM_LAHIRI             1
#define SE_SIDM_DELUCE             2
#define SE_SIDM_RAMAN              3
#define SE_SIDM_USHASHASHI         4
#define SE_SIDM_KRISHNAMURTI       5
#define SE_SIDM_DJWHAL_KHUL        6
#define SE_SIDM_YUKTESHWAR         7
#define SE_SIDM_JN_BHASIN          8
#define SE_SIDM_BABYL_KUGLER1      9
#define SE_SIDM_BABYL_KUGLER2      10
#define SE_SIDM_BABYL_KUGLER3      11
#define SE_SIDM_BABYL_HUBER        12
#define SE_SIDM_BABYL_ETPSC        13
#define SE_SIDM_ALDEBARAN_15TAU    14
#define SE_SIDM_HIPPARCHOS         15
#define SE_SIDM_SASSANIAN          16
#define SE_SIDM_GALCENT_0SAG       17
#define SE_SIDM_J2000              18
#define SE_SIDM_J1900              19
#define SE_SIDM_B1950              20
#define SE_SIDM_SURYASIDDHANTA     21
#define SE_SIDM_SURYASIDDHANTA_MSUN 22
#define SE_SIDM_ARYABHATA          23
#define SE_SIDM_ARYABHATA_MSUN     24
#define SE_SIDM_USER               255
```

For information about the sidereal modes, read the chapter on sidereal calculations in swisseph.doc.
To define your own sidereal mode, use SE_SIDM_USER (= 255) and set the reference date **(t0)** and the initial value of the **ayanamsha (ayan_t0)**.

```
ayan_t0 = tropical_position_t0 – sidereal_position_t0.
```

Without additional specifications, the traditional method is used. The **ayanamsha** measured on the ecliptic of t0 is subtracted from tropical positions referred to the ecliptic of date.

Note, this method will NOT provide accurate results if you want coordinates referred to the ecliptic of one of the following equinoxes:

```
#define SE_SIDM_J2000              18
#define SE_SIDM_J1900              19
#define SE_SIDM_B1950              20
```

Instead, you have to use a correct coordinate transformation as described in the following:

Special uses of the sidereal functions:

a) correct transformation of ecliptic coordinates to the ecliptic of a particular date

If a correct transformation to the ecliptic of **t0** is required the following bit can be added ('ored') to the value of the variable **sid_mode:**

```
/* for projection onto ecliptic of t0 */
#define SE_SIDBIT_ECL_T0      256
```

E.g.:

```
swe_set_sid_mode(SE_SIDM_J2000 + SE_SIDBIT_ECL_T0, 0, 0);
iflag |= SEFLG_SIDEREAL;
for (i = 0; i < NPLANETS; i++) {
  iflgret = swe_calc(tjd, ipl, iflag, xp, serr);
  printf("%f\n", xp[0]);
}
```

This procedure is required for the following sidereal modes, i.e. for transformation to the ecliptic of one of the standard equinoxes:

```
#define SE_SIDM_J2000              18
#define SE_SIDM_J1900              19
#define SE_SIDM_B1950              20
```

b) calculating precession-corrected transits

The function swe_set_sidmode() can also be used for calculating ”precession-corrected transits”. There are two methods, of which you have to choose the one that is more appropriate for you:

1. If you already have tropical positions of a natal chart, you can proceed as follows:

```
iflgret = swe_calc(tjd_et_natal, SE_ECL_NUT, 0, x, serr);
nut_long_nata = x[2];
swe_set_sid_mode( SE_SIDBIT_USER + SE_SIDBIT_ECL_T0, tjd_et, nut_long_natal );
```

where **tjd_et_natal** is the Julian day of the natal chart (Ephemeris time).
After this calculate the transits, using the function swe_calc() with the sidereal bit:

```
iflag |= SEFLG_SIDEREAL;
iflgret = swe_calc(tjd_et_transit, ipl_transit, iflag, xpt, serr);
```

2. If you do not have tropical natal positions yet, if you do not need them and are just interested in transit times, you can have it simpler:

```
swe_set_sid_mode( SE_SIDBIT_USER + SE_SIDBIT_ECL_T0, tjd_et, 0 );
iflag |= SEFLG_SIDEREAL;
iflgret = swe_calc(tjd_et_natal, ipl_natal, iflag, xp, serr);
iflgret = swe_calc(tjd_et_transit, ipl_transit, iflag, xpt, serr);
```

In this case, the natal positions will be tropical but without nutation. Note that you should not use them for other purposes.

c) solar system rotation plane

For sidereal positions referred to the solar system rotation plane, use the flag

```
/* for projection onto solar system rotation plane */
#define SE_SIDBIT_SSY_PLANE     512
```

Note: the parameters set by swe_set_sid_mode() survive calls of the function swe_close().

10.2. swe_get_ayanamsa_ut() and swe_get_ayanamsa()

```
double swe_get_ayanamsa_ut(double tjd_ut);
double swe_get_ayanamsa(double tjd_et);
```

The function swe_get_ayanamsa_ut() was introduced with Swisseph Version 1.60 and expects Universal Time instead of Ephemeris Time. (cf. swe_calc_ut() and swe_calc())
The two functions compute the **ayanamsha**, i.e. the distance of the tropical vernal point from the sidereal zero point of the zodiac. The **ayanamsha** is used to compute sidereal planetary positions from tropical ones:

```
pos_sid = pos_trop – ayanamsha
```

Before calling swe_get_ayanamsha(), you have to set the sidereal mode with swe_set_sid_mode, unless you want the default sidereal mode, which is the Fagan/Bradley **ayanamsha.**

11. The Ephemeris file related functions

11.1 swe_set_ephe_path()

If the environment variable SE_EPHE_PATH exists in the environment where Swiss Ephemeris is used, its content is used to find the ephemeris files. The variable can contain a directory name, or a list of directory names separated by **;** (semicolon) on Windows or **:** (colon) on Unix.

```
int swe_set_ephe_path(char *path);
```

Usually an application will want to set its own ephemeris path by calling swe_set_ephe_path(), e.g.

```
swe_set_ephe_path(”C:\\SWEPH\\EPHE”);
```

The argument can be a single directory name or a list of directories, which are then searched in sequence. The argument of this call is ignored if the environment variable SE_EPHE_PATH exists and is not empty.

If you want to make sure that your program overrides any environment variable setting, you can use putenv() to set it to an empty string.

If the path is longer than **256 bytes**, swe_set_ephe_path() sets the path \SWEPH\EPHE instead.
If no environment variable exists and swe_set_ephe_path() is never called, the built-in ephemeris path is used. On Windows it is "\sweph\ephe" relative to the current working drive, on Unix it is "/users/ephe".
Asteroid ephemerides are looked for in the subdirectories ast0, ast1, ast2 .. ast9 of the ephemeris directory and, if not found there, in the ephemeris directory itself. Asteroids with numbers 0 – 999 are expected in directory ast0, those with numbers 1000 – 1999 in directory ast1 etc.
The environment variable SE_EPHE_PATH is most convenient when a user has several applications installed which all use the Swiss Ephemeris but would normally expect the ephemeris files in different application-specific directories. The use can override this by setting the environment variable, which forces all the different applications to use the same ephemeris directory. This allows him to use only one set of installed ephemeris files for all different applications. A developer should accept this override feature and allow the sophisticated users to exploit it.

11.2 swe_close()

```
/* close Swiss Ephemeris */
void swe_close(void);
```

At the end of your computations you can release most resources (open files and allocated memory) used by the Swiss Ephemeris DLL.
The following parameters survive a call of swe_calc():

- the ephemeris path set by swe_set_ephe_path()
- the JPL file name set by swe_set_jpl_file()
- the geographical location set by swe_set_topo() for topocentric planetary positions
- the sidereal mode set by swe_set_sid_mode() for sidereal planetary positions

As soon as you make a call to swe_calc() or swe_fixstar(), the Swiss Ephemeris re-opens again.

11.3 swe_set_jpl_file()

```
/* set name of JPL ephemeris file */
int swe_set_jpl_file(char *fname);
```

If you work with the JPL ephemeris, SwissEph uses the default file name which is defined in swephexp.h as SE_FNAME_DFT. Currently, it has the value "de406.eph".
If different JPL ephemeris file is required, call the function swe_set_jpl_file() to make the file name known to the software, e.g.

```
swe_set_jpl_file("de405.eph");
```

This file must reside in the ephemeris path you are using for all your ephemeris files.
If the file name is longer than 256 byte, swe_set_jpl_file() cuts the file name to a length of 256 bytes. The error will become visible after the first call of swe_calc(), when it will return zero positions and an error message.

11.4 swe_version()

```
/* find out version number of your Swiss Ephemeris version */
char *swe_version(char *svers);
/* svers is a string variable with sufficient space to contain the version number (255 char) */
```

The Function returns a pointer to the string svers, i.e. to the version number of the Swiss Ephemeris that your software is using.

12. House cusp calculation

12.1 swe_houses()

```
/* house cusps, ascendant and MC */
int swe_houses(
double tjd_ut,          /* Julian day number, UT */
double geolat,          /* geographic latitude, in degrees */
double geolon,          /* geographic longitude, in degrees
                         * eastern longitude is positive,
                         * western longitude is negative,
                         * northern latitude is positive,
                         * southern latitude is negative */
int hsys,               /* house method, ascii code of one of the letters PKORCAEVXHTBG */
double *cusps,          /* array for 13 doubles */
double *ascmc);         /* array for 10 doubles */
```

12.2 swe_houses_armc()

```
int swe_houses_armc(
double armc,            /* ARMC */
double geolat,          /* geographic latitude, in degrees */
double eps,             /* ecliptic obliquity, in degrees */
int hsys,               /* house method, ascii code of one of the letters PKORCAEVXHTBG */
double *cusps,          /* array for 13 doubles */
double *ascmc);         /* array for 10 doubles */
```

12.3 swe_houses_ex()

```
/* extended function; to compute tropical or sidereal positions */
int swe_houses_ex(
      double tjd_ut,    /* Julian day number, UT */
      int32 iflag,      /* 0 or SEFLG_SIDEREAL or SEFLG_RADIANS */
      double geolat,    /* geographic latitude, in degrees */
      double geolon,    /* geographic longitude, in degrees
                         * eastern longitude is positive,
                         * western longitude is negative,
                         * northern latitude is positive,
                         * southern latitude is negative */
      int hsys,         /* house method, ascii code of one of the letters PKORCAEVXHTBG */
      double *cusps,    /* array for 13 doubles */
double *ascmc);         /* array for 10 doubles */
```

The function swe_houses() is most comfortable, if you need the houses for a given date and geographic position. Sometimes, however, you will want to compute houses from an ARMC, e.g. with the composite horoscope which has no date, only the composite ARMC of two natal ARMCs. In such cases, you can use the function swe_houses_armc(). To compute the composite ecliptic obliquity **eps**, you will have to call sweph_calc() with **ipl** = SE_ECL_NUT for both birth dates and calculate the average of both **eps.**

Note that **tjd_ut** must be Universal Time, whereas planets are computed from Ephemeris Time

tjd_et = tjd_ut + delta_t(tjd_ut).

Also note that the array **cusps** must provide space for **13 doubles** (declare as **cusp[13]**), otherwise you risk a program crash. With house system 'G' (Gauquelin sector cusps), declare it as **cusp[37]**.

Note: With house system 'G', the cusp numbering is in clockwise direction.

The extended house function swe_houses_ex() does exactly the same calculations as swe_houses(). The difference is that swe_houses_ex() has a parameter **iflag**, which can be set to SEFLG_SIDEREAL, if sidereal house positions are wanted. Before calling swe_houses_ex() for sidereal house positions, the sidereal mode can be set by calling the function swe_set_sid_mode(). If this is not done, the default sidereal mode, i.e. the Fagan/Bradley ayanamsha, will be used.

There is no extended function for swe_houses_armc(). Therefore, if you want to compute such obscure things as sidereal composite house cusps, the procedure will be more complicated:

```
/* sidereal composite house computation; with true epsilon, but without nutation in longitude */
swe_calc(tjd_et1, SE_ECL_NUT, 0, x1, serr);
swe_calc(tjd_et2, SE_ECL_NUT, 0, x2, serr);
armc1 = swe_sidtime(tjd_ut1) * 15;
armc2 = swe_sidtime(tjd_ut2) * 15;
armc_comp = composite(armc1, armc2); /* this is a function created by the user */
eps_comp = (x1[0] + x2[0]) / 2;
nut_comp = (x1[2] + x2[2]) / 2;
tjd_comp = (tjd_et1 + tjd_et2) / 2;
aya = swe_get_ayanamsa(tjd_comp);
swe_houses_armc(armc_comp, geolat, eps_comp, hsys, cusps, ascmc);
for (i = 1; i <= 12; i++)
  cusp[i] = swe_degnorm(cusp[i] – aya – nut_comp);
for (i = 0; i < 10; i++)
  ascmc[i] = swe_degnorm(asc_mc[i] – aya – nut_comp);
```

Output and input parameters.
The first array element **cusps[0]** is always 0, the twelve houses follow in **cusps[1] .. [12]**, the reason being that arrays in C begin with the index 0. The indices are therefore:

```
cusps[0] = 0
cusps[1] = house 1
cusps[2] = house 2
```

etc.

In the array **ascmc,** the function returns the following values:

```
ascmc[0] =          Ascendant
ascmc[1] =          MC
ascmc[2] =          ARMC
ascmc[3] =          Vertex
ascmc[4] =          "equatorial ascendant"
ascmc[5] =          "co-ascendant" (Walter Koch)
ascmc[6] =          "co-ascendant" (Michael Munkasey)
ascmc[7] =          "polar ascendant" (M. Munkasey)
```

The following defines can be used to find these values:

```
#define SE_ASC          0
#define SE_MC           1
#define SE_ARMC         2
#define SE_VERTEX       3
#define SE_EQUASC       4       /* "equatorial ascendant" */
#define SE_COASC1       5       /* "co-ascendant" (W. Koch) */
#define SE_COASC2       6       /* "co-ascendant" (M. Munkasey) */
#define SE_POLASC       7       /* "polar ascendant" (M. Munkasey) */
#define SE_NASCMC       8
```

ascmc must be an array of **10 doubles**. **ascmc[8... 9]** are 0 and may be used for additional points in future releases.

The following house systems are implemented so far

```
hsys =      'P'         Placidus
            'K'         Koch
            'O'         Porphyrius
            'R'         Regiomontanus
            'C'         Campanus
            'A' or 'E'  Equal (cusp 1 is Ascendant)
            'V'         Vehlow equal (Asc. in middle of house 1)
            'W'         Whole sign
            'X'         axial rotation system / meridian system / zariel
            'H'         azimuthal or horizontal system
            'T'         Polich/Page ("topocentric" system)
            'B'         Alcabitus
            'M'         Morinus
            'U'         Krusinski-Pisa
            'G'         Gauquelin sectors
```

Placidus and Koch house cusps **cannot be computed beyond the polar circle**. In such cases, swe_houses() switches to Porphyry houses (each quadrant is divided into three equal parts) and returns the error code ERR.

The **Vertex** is the point on the ecliptic that is located in precise western direction. The opposition of the **Vertex** is the **Antivertex,** the ecliptic east point.

13. The sign of geographical longitudes in Swisseph functions

There is a disagreement between American and **European** programmers whether eastern or western geographical longitudes ought to be considered positive. Americans prefer to have West longitudes positive, Europeans prefer the older tradition that considers East longitudes as positive and West longitudes as negative. The Astronomical Almanac still follows the European pattern. It gives the geographical coordinates of observatories in "East longitude".
The Swiss Ephemeris also follows the European style. All Swiss Ephemeris functions that use geographical coordinates consider **positive geographical longitudes as East** and **negative ones as West**.
E.g. 87w39 = -87.65° (Chicago IL/USA) and 8e33 = +8.55° (Zurich, Switzerland).
There is no such controversy about northern and southern geographical latitudes. North is always positive and south is negative.

14. Getting the house position of a planet with swe_house_pos()

To compute the house position of a given body for a given ARMC, you may use the

```
double swe_house_pos(
        double armc,        /* ARMC */
        double geolat,      /* geographic latitude, in degrees */
        double eps,         /* ecliptic obliquity, in degrees */
        int hsys,           /* house method, one of the letters PKRCAV */
        double *xpin,       /* array of 2 doubles: ecl. longitude and latitude of the planet */
char *serr);                /* return area for error or warning message */
```

The variables **armc, geolat, eps,** and **xpin[0]** and **xpin[1]** (ecliptic longitude and latitude of the planet) must be in degrees. **serr** must, as usually, point to a character array of 256 byte.
The function returns a value between 1.0 and 12.999999, indicating in which house a planet is and how far from its cusp it is.
With house system 'G' (Gauquelin sectors), a value between 1.0 and 36.9999999 is returned. Note that, while all other house systems number house cusps in counterclockwise direction, Gauquelin sectors are numbered in clockwise direction.

With Koch houses, the function sometimes returns 0, if the computation was not possible. This happens most often in polar regions, but it can happen at latitudes **below 66°33'** as well, e.g. if a body has a high declination and falls within the circumpolar sky. With circumpolar fixed stars (or asteroids) a Koch house position may be impossible at any geographic location except on the equator.
The user must decide how to deal with this situation.
You can use the house positions returned by this function for house horoscopes (or "mundane" positions). For this, you have to transform it into a value between 0 and 360 degrees. Subtract 1 from the house number and multiply it with 30, or mund_pos = (hpos – 1) * 30;

You will realize that house positions computed like this, e.g. for the Koch houses, will not agree exactly with the ones that you get applying the Huber "hand calculation" method. If you want a better agreement, set the ecliptic latitude **xpin[1]= 0**;. Remaining differences result from the fact that Huber's hand calculation is a simplification, whereas our computation is geometrically accurate.

This function requires TROPICAL positions in **xpin**. SIDEREAL house positions are identical to tropical ones in the following cases:

- If the traditional method is used to compute sidereal planets (sid_pos = trop_pos – ayanamsha). Here the function swe_house_pos() works for all house systems.
- If a non-traditional method (projection to the ecliptic of t0 or to the solar system rotation plane) is used and the definition of the house system does not depend on the ecliptic. This is the case with Campanus, Regiomontanus, Placidus, Azimuth houses, axial rotation houses. This is NOT the case with equal houses, Porphyry and Koch houses. You have to compute equal and Porphyry house positions on your own. **We recommend to avoid Koch** houses here. Sidereal Koch houses make no sense with these sidereal algorithms.

- Alcabitus is not yet supported in release 1.61.01

14.1. Calculating the Gauquelin sector position of a planet with swe_house_pos() or swe_gauquelin_sector()

For general information on Gauquelin sectors, read chapter 6.5 in documentation file swisseph.doc.

There are two functions that can be used to calculate Gauquelin sectors:

- swe_house_pos. Full details about this function are presented in the previous section. To calculate Gauquelin sectors the parameter hsys must be set to 'G' (Gauquelin sectors). This function will then return the sector position as a value between 1.0 and 36.9999999. Note that Gauquelin sectors are numbered in clockwise direction, unlike all other house systems.
- swe_gauquelin_sector - detailed below.

Function swe_gauquelin_sector() is declared as follows:

```
int32 swe_gauquelin_sector(
double tjd_ut,          /* search after this time (UT) */
int32 ipl,              /* planet number, if planet, or moon */
char *starname,         /* star name, if star */
int32 iflag,            /* flag for ephemeris and SEFLG_TOPOCTR */
int32 imeth,            /* method: 0 = with lat., 1 = without lat.,
                        /*         2 = from rise/set, 3 = from rise/set with refraction */
double *geopos,         /* array of three doubles containing
                         * geograph. long., lat., height of observer */
double atpress,         /* atmospheric pressure, only useful with imeth=3;
                          * if 0, default = 1013.25 mbar is used*/
double attemp,          /* atmospheric temperature in degrees Celsius, only useful with imeth=3 */
double *dgsect,         /* return address for gauquelin sector position */
char *serr);            /* return address for error message */
```

This function returns OK or ERR (-1). It returns an error in a number of cases, for example circumpolar bodies with imeth=2. As with other SE functions, if there is an error, an error message is written to serr. dgsect is used to obtain the Gauquelin sector sector position as a value between 1.0 and 36.9999999. Gauquelin sectors are numbered in clockwise direction.

There are six methods of computing the Gauquelin sector position of a planet:

1. Sector positions from ecliptical longitude AND latitude:
   There are two ways of doing this:
   - Call swe_house_pos() with hsys = 'G', xpin[0] = ecliptical longitude of planet, and xpin[1] = ecliptical latitude. This function returns the sector position as a value between 1.0 and 36.9999999.
   - Call swe_gauquelin_sector() with imeth=0. This is less efficient than swe_house_pos because it recalculates the whole planet whereas swe_house_pos() has an input array for ecliptical positions calculated before.

2. Sector positions computed from ecliptical longitudes without ecliptical latitudes:
   There are two ways of doing this:
   - Call swe_house_pos() with hsys = 'G', xpin[0] = ecl. longitude of planet, and xpin[1] = 0. This function returns the sector position as a value between 1.0 and 36.9999999.
   - Call swe_gauquelin_sector() with imeth=1. Again this is less efficient than swe_house_pos.

3. Sector positions of a planet from rising and setting times of planets.
   The rising and setting of the disk center is used:
   - Call swe_gauquelin_sector() with imeth=2.

4. Sector positions of a planet from rising and setting times of planets, taking into account atmospheric refraction.
   The rising and setting of the disk center is used:
   - Call swe_gauquelin_sector() with imeth = 3.

5. Sector positions of a planet from rising and setting times of planets.
   The rising and setting of the disk edge is used:
   - Call swe_gauquelin_sector() with imeth=4.

6. Sector positions of a planet from rising and setting times of planets, taking into account atmospheric refraction.

The rising and setting of the disk edge is used:

- Call swe_gauquelin_sector() with imeth = 5.

15. Sidereal time with swe_sidtime() and swe_sidtime0()

The sidereal time is computed inside the houses() function and returned via the variable **armc** which measures sidereal time in degrees. To get sidereal time in hours, divide **armc** by 15.
If the sidereal time is required separately from house calculation, two functions are available. The second version requires obliquity and nutation to be given in the function call, the first function computes them internally. Both return sidereal time at the Greenwich Meridian, measured in hours.

```
double swe_sidtime(double tjd_ut);    /* Julian day number, UT */
double swe_sidtime0(
      double tjd_ut,      /* Julian day number, UT */
      double eps,         /* obliquity of ecliptic, in degrees */
      double nut);        /* nutation in longitude, in degrees */
```

16. Summary of SWISSEPH functions

16.1. Calculation of planets and stars

Planets, moon, asteroids, lunar nodes, apogees, fictitious bodies

```
long swe_calc_ut(
        double tjd_ut,      /* Julian day number, Universal Time */
        int ipl,            /* planet number */
        long iflag,         /* flag bits */
        double *xx,         /* target address for 6 position values: longitude, latitude, distance,
                               long. speed, lat. speed, dist. speed */
        char *serr);        /* 256 bytes for error string */

long swe_calc(
        double tjd_et,      /* Julian day number, Ephemeris Time */
        int ipl,            /* planet number */
        long iflag,         /* flag bits */
        double *xx,         /* target address for 6 position values: longitude, latitude, distance,
                               long. speed, lat. speed, dist. speed */
        char *serr);        /* 256 bytes for error string */
```

Fixed stars

```
long swe_fixstar_ut(
        char *star,         /* star name, returned star name 40 bytes */
        double tjd_ut,      /* Julian day number, Universal Time */
        long iflag,         /* flag bits */
        double *xx,         /* target address for 6 position values: longitude, latitude, distance,
                                   long. speed, lat. speed, dist. speed */
        char *serr);        /* 256 bytes for error string */

long swe_fixstar(
        char *star,         /* star name, returned star name 40 bytes */
        double tjd_et,      /* Julian day number, Ephemeris Time */
        long iflag,         /* flag bits */
        double *xx,         /* target address for 6 position values: longitude, latitude, distance,
                               long. speed, lat. speed, dist. speed */
        char *serr);        /* 256 bytes for error string */
```

Set the geographic location for topocentric planet computation

```
void swe_set_topo (
        double geolon,      /* geographic longitude */
        double geolat,      /* geographic latitude
                               eastern longitude is positive,
                               western longitude is negative,
                               northern latitude is positive,
                               southern latitude is negative */
        double altitude);   /* altitude above sea */
```

Set the sidereal mode for sidereal planet positions

```
void swe_set_sid_mode (
        int32 sid_mode,
        double t0,          /* reference epoch */
        double ayan_t0);    /* initial ayanamsha at t0 */

/* to get the ayanamsha for a date */
double swe_get_ayanamsa(double tjd_et);
```

16.2 Eclipses and planetary phenomena

Find the next eclipse for a given geographic position

```
int32 swe_sol_eclipse_when_loc(
double tjd_start,       /* start date for search, Jul. day UT */
int32 ifl,              /* ephemeris flag */
double *geopos,         /* 3 doubles for geo. lon, lat, height */
                        * eastern longitude is positive,
                        * western longitude is negative,
                        * northern latitude is positive,
                        * southern latitude is negative */
double *tret,           /* return array, 10 doubles, see below */
double *attr,           /* return array, 20 doubles, see below */
AS_BOOL backward,       /* TRUE, if backward search */
char *serr);            /* return error string */
```

Find the next eclipse globally

```
int32 swe_sol_eclipse_when_glob(
double tjd_start,       /* start date for search, Jul. day UT */
int32 ifl,              /* ephemeris flag */
int32 ifltype,          /* eclipse type wanted: SE_ECL_TOTAL etc. */
double *tret,           /* return array, 10 doubles, see below */
AS_BOOL backward,       /* TRUE, if backward search */
char *serr);            /* return error string */
```

Compute the attributes of a solar eclipse for a given tjd, geographic long., latit. and height

```
int32 swe_sol_eclipse_how(
double tjd_ut,          /* time, Jul. day UT */
int32 ifl,              /* ephemeris flag */
double *geopos,         /* geogr. longitude, latitude, height */
                        * eastern longitude is positive,
                        * western longitude is negative,
                        * northern latitude is positive,
                        * southern latitude is negative */
double *attr,           /* return array, 20 doubles, see below */
char *serr);            /* return error string */
```

Find out the geographic position where a central eclipse is central or a non-central one maximal

```
int32 swe_sol_eclipse_where (
double tjd_ut,          /* time, Jul. day UT */
int32 ifl,                      /* ephemeris flag */
double *geopos,         /* return array, 2 doubles, geo. long. and lat. */
                        * eastern longitude is positive,
                        * western longitude is negative,
                        * northern latitude is positive,
                        * southern latitude is negative */
double *attr,           /* return array, 20 doubles, see below */
char *serr);            /* return error string */

or

int32 swe_lun_occult_where (
double tjd_ut,          /* time, Jul. day UT */
int32 ipl,            /* planet number */
char* starname,       /* star name, must be NULL or "" if not a star */
int32 ifl,              /* ephemeris flag */
double *geopos,         /* return array, 2 doubles, geo. long. and lat.
                        * eastern longitude is positive,
                        * western longitude is negative,
                        * northern latitude is positive,
                        * southern latitude is negative */
double *attr,           /* return array, 20 doubles, see below */
char *serr);            /* return error string */
```

Find the next occultation of a body by the moon for a given geographic position

(can also be used for solar eclipses)

```
int32 swe_lun_occult_when_loc(
double tjd_start,      /* start date for search, Jul. day UT */
int32 ipl,             /* planet number */
char* starname,        /* star name, must be NULL or "" if not a star */
int32 ifl,             /* ephemeris flag */
double *geopos,        /* 3 doubles for geo. lon, lat, height eastern longitude is positive,
                          western longitude is negative,  northern latitude is positive,
                          southern latitude is negative */
double *tret,          /* return array, 10 doubles, see below */
double *attr,          /* return array, 20 doubles, see below */
AS_BOOL backward,      /* TRUE, if backward search */
char *serr);           /* return error string */
```

Find the next occultation globally

(can also be used for solar eclipses)

```
int32 swe_lun_occult_when_glob(
double tjd_start,      /* start date for search, Jul. day UT */
int32 ipl,             /* planet number */
char* starname,        /* star name, must be NULL or "" if not a star */
int32 ifl,             /* ephemeris flag */
int32 ifltype,         /* eclipse type wanted */
double *geopos,        /* 3 doubles for geo. lon, lat, height eastern longitude is positive,
                          western longitude is negative,  northern latitude is positive,
                          southern latitude is negative */
double *tret,          /* return array, 10 doubles, see below */
double *attr,          /* return array, 20 doubles, see below */
AS_BOOL backward,      /* TRUE, if backward search */
char *serr);           /* return error string */
```

Find the next lunar eclipse

```
int32 swe_lun_eclipse_when(
double tjd_start,      /* start date for search, Jul. day UT */
int32 ifl,                     /* ephemeris flag */
int32 ifltype,         /* eclipse type wanted: SE_ECL_TOTAL etc. */
double *tret,          /* return array, 10 doubles, see below */
AS_BOOL backward,      /* TRUE, if backward search */
char *serr);           /* return error string */
```

Compute the attributes of a lunar eclipse at a given time

```
int32 swe_lun_eclipse_how(
double tjd_ut,         /* time, Jul. day UT */
int32 ifl,             /* ephemeris flag */
double *geopos,        /* input array, geopos, geolon, geoheight */
                        eastern longitude is positive,
                        western longitude is negative,
                        northern latitude is positive,
                        southern latitude is negative */
double *attr,          /* return array, 20 doubles, see below */
char *serr);           /* return error string */
```

Compute risings, settings and meridian transits of a body

```
int32 swe_rise_trans(
double tjd_ut,         /* search after this time (UT) */
int32 ipl,             /* planet number, if planet or moon */
char *starname,        /* star name, if star */
int32 epheflag,        /* ephemeris flag */
int32 rsmi,            /* integer specifying that rise, set, or one of the two meridian transits is
                          wanted. see definition below */
double *geopos,        /* array of three doubles containing geograph. long., lat., height of observer */
```

```
double atpress,        /* atmospheric pressure in mbar/hPa */
double attemp,         /* atmospheric temperature in deg. C */
double *tret,          /* return address (double) for rise time etc. */
char *serr);           /* return address for error message */

int32 swe_rise_trans_true_hor(
double tjd_ut,         /* search after this time (UT) */
int32 ipl,             /* planet number, if planet or moon */
char *starname,        /* star name, if star */
int32 epheflag,        /* ephemeris flag */
int32 rsmi,            /* integer specifying that rise, set, orone of the two meridian transits is
                          wanted. see definition below */
double *geopos,        /* array of three doubles containing
                          * geograph. long., lat., height of observer */
double atpress,        /* atmospheric pressure in mbar/hPa */
double attemp,         /* atmospheric temperature in deg. C */
double horhgt,         /* height of local horizon in deg at the point where the body rises or sets*/
double *tret,          /* return address (double) for rise time etc. */
char *serr);           /* return address for error message */
```

Compute planetary phenomena

```
int32 swe_pheno_ut(
double tjd_ut,         /* time Jul. Day UT */
int32 ipl,             /* planet number */
int32 iflag,           /* ephemeris flag */
double *attr,          /* return array, 20 doubles, see below */
char *serr);           /* return error string */
int32 swe_pheno(
double tjd_et,         /* time Jul. Day ET */
int32 ipl,             /* planet number */
int32 iflag,           /* ephemeris flag */
double *attr,          /* return array, 20 doubles, see below */
char *serr);           /* return error string */

void swe_azalt(
    double tjd_ut,     /* UT */
    int32 calc_flag,   /* SE_ECL2HOR or SE_EQU2HOR */
    double *geopos,    /* array of 3 doubles: geogr. long., lat., height */
    double atpress,    /* atmospheric pressure in mbar (hPa) */
    double attemp,     /* atmospheric temperature in degrees Celsius */
    double *xin,       /* array of 3 doubles: position of body in either  ecliptical or equatorial
                          coordinates,  depending on calc_flag */
    double *xaz);      /* return array of 3 doubles, containing azimuth, true altitude, apparent
                          altitude */

void swe_azalt_rev(
    double tjd_ut,
    int32 calc_flag,   /* either SE_HOR2ECL or SE_HOR2EQU */
    double *geopos,    /* array of 3 doubles for geograph. pos. of observer */
    double *xin,       /* array of 2 doubles for azimuth and true altitude of planet */
    double *xout);     /* return array of 2 doubles for either ecliptic or equatorial coordinates,
                          depending on calc_flag */
double swe_refrac(
double inalt,
double atpress,        /* atmospheric pressure in mbar (hPa) */
double attemp,         /* atmospheric temperature in degrees Celsius */
int32 calc_flag);      /* either SE_TRUE_TO_APP or SE_APP_TO_TRUE */
```

16.3. Date and time conversion

Delta T from Julian day number

```
* Ephemeris time (ET) = Universal time (UT) + swe_deltat(UT)*/
```

```
double swe_deltat(double tjd);
```

Julian day number from year, month, day, hour, with check whether date is legal

```
/*Return value: OK or ERR  */
int swe_date_conversion (
      int y , int m , int d ,        /* year, month, day */
      double hour,                   /* hours (decimal, with fraction) */
      char c,                        /* calendar 'g'[regorian]|'j'[ulian] */
double *tjd);                        /* target address for Julian day */
```

Julian day number from year, month, day, hour

```
double swe_julday(
int year, int month, int day, double hour,
int gregflag);              /* Gregorian calendar: 1, Julian calendar: 0 */
```

Year, month, day, hour from Julian day number

```
void swe_revjul (
      double tjd,          /* Julian day number */
      int gregflag,        /* Gregorian calendar: 1, Julian calendar: 0 */
      int *year,           /* target addresses for year, etc. */
      int *month, int *day, double *hour);
```

Local time to UTC and UTC to local time

```
/* transform local time to UTC or UTC to local time
 *
 * input:
 *   iyear ... dsec     date and time
 *   d_timezone         timezone offset
 * output:
 *   iyear_out ... dsec_out
 *
 * For time zones east of Greenwich, d_timezone is positive.
 * For time zones west of Greenwich, d_timezone is negative.
 *
 * For conversion from local time to utc, use +d_timezone.
 * For conversion from utc to local time, use -d_timezone.
 */
void FAR PASCAL_CONV swe_utc_timezone(
      int32 iyear, int32 imonth, int32 iday,
      int32 ihour, int32 imin, double dsec,
      double d_timezone,
      int32 *iyear_out, int32 *imonth_out, int32 *iday_out,
      int32 *ihour_out, int32 *imin_out, double *dsec_out
      )
```

UTC to jd (TT and UT1)

```
/* input: date and time (wall clock time), calendar flag.
 * output: an array of doubles with Julian Day number in ET (TT) and UT (UT1)
 *          an error message (on error)
 * The function returns OK or ERR.
 */
void swe_utc_to_jd (
      int32 iyear, int32 imonth, int32 iday,
      int32 ihour, int32 imin, double dsec,   /* note : second is a decimal */
      gregflag,            /* Gregorian calendar: 1, Julian calendar: 0 */
      dret                 /* return array, two doubles:
                            * dret[0] = Julian day in ET (TT)
                            * dret[1] = Julian day in UT (UT1) */
```

```
        serr                /* error string */
)
```

TT (ET1) to UTC

```
/* input: Julian day number in ET (TT), calendar flag
 * output: year, month, day, hour, min, sec in UTC */
void swe_jdet_to_utc (
        double tjd_et,      /* Julian day number in ET (TT) */
        gregflag,           /* Gregorian calendar: 1, Julian calendar: 0 */
        int32 *iyear, int32 *imonth, int32 *iday,
        int32 *ihour, int32 *imin, double *dsec,   /* note : second is a decimal */
)
```

UTC to TT (ET1)

```
/* input: Julian day number in UT (UT1), calendar flag
 * output: year, month, day, hour, min, sec in UTC */
void swe_jdut1_to_utc (
        double tjd_ut,      /* Julian day number in ET (TT) */
        gregflag,           /* Gregorian calendar: 1, Julian calendar: 0 */
        int32 *iyear, int32 *imonth, int32 *iday,
        int32 *ihour, int32 *imin, double *dsec,   /* note : second is a decimal */
)
```

Get tidal acceleration used in swe_deltat()

```
double swe_get_tid_acc(void);
```

Set tidal acceleration to be used in swe_deltat()

```
void swe_set_tid_acc(double t_acc);
```

Equation of time

```
/ * function returns the difference between local apparent and local mean time.
e = LAT – LMT.  tjd_et is ephemeris time */
int swe_time_equ(double tjd_et, double *e, char *serr);
```

16.4. Initialization, setup, and closing functions

Set directory path of ephemeris files

```
int swe_set_ephe_path(char *path);

/* set name of JPL ephemeris file */
int swe_set_jpl_file(char *fname);

/* close Swiss Ephemeris */
void swe_close(void);
```

16.5. House calculation

Sidereal time

```
double swe_sidtime(double tjd_ut);    /* Julian day number, UT */

double swe_sidtime0(
        double tjd_ut,      /* Julian day number, UT */
        double eps,         /* obliquity of ecliptic, in degrees */
        double nut);        /* nutation, in degrees */
```

House cusps, ascendant and MC

```
int swe_houses(
        double tjd_ut,      /* Julian day number, UT */
        double geolat,      /* geographic latitude, in degrees */
        double geolon,      /* geographic longitude, in degrees
                               eastern longitude is positive,
                               western longitude is negative,
                               northern latitude is positive,
                               southern latitude is negative */
        int hsys,           /* house method, one of the letters PKRCAV */
        double* cusps,      /* array for 13 doubles */
        double* ascmc);     /* array for 10 doubles */
```

Extended house function; to compute tropical or sidereal positions

```
int swe_houses_ex(
        double tjd_ut,      /* Julian day number, UT */
        int32 iflag,        /* 0 or SEFLG_SIDEREAL or SEFLG_RADIANS */
        double geolat,      /* geographic latitude, in degrees */
        double geolon,      /* geographic longitude, in degrees
                               eastern longitude is positive,
                               western longitude is negative,
                               northern latitude is positive,
                               southern latitude is negative */
        int hsys,           /* house method, one of the letters PKRCAV */
        double* cusps,      /* array for 13 doubles */
        double* ascmc);     /* array for 10 doubles */

int swe_houses_armc(
        double armc,        /* ARMC */
        double geolat,      /* geographic latitude, in degrees */
        double eps,         /* ecliptic obliquity, in degrees */
        int hsys,           /* house method, one of the letters PKRCAV */
        double *cusps,      /* array for 13 doubles */
        double *ascmc);     /* array for 10 doubles */
```

Get the house position of a celestial point

```
double swe_house_pos (
        double armc,        /* ARMC */
        double geolat,      /* geographic latitude, in degrees
                               eastern longitude is positive,
                               western longitude is negative,
                               northern latitude is positive,
                               southern latitude is negative */
        double eps,         /* ecliptic obliquity, in degrees */
        int hsys,           /* house method, one of the letters PKRCAV */
        double *xpin,       /* array of 2 doubles: ecl. longitude and latitude of the planet */
        char *serr);        /* return area for error or warning message */
```

Get the Gauquelin sector position for a body

```
double swe_gauquelin_sector(
double tjd_ut,          /* search after this time (UT) */
int32 ipl,              /* planet number, if planet, or moon */
char *starname,         /* star name, if star */
int32 iflag,            /* flag for ephemeris and SEFLG_TOPOCTR */
int32 imeth,            /* method: 0 = with lat., 1 = without lat.,
                        /*           2 = from rise/set, 3 = from rise/set with refraction */
double *geopos,         /* array of three doubles containing
                         * geograph. long., lat., height of observer */
double atpress,         /* atmospheric pressure, only useful with imeth=3;
                          * if 0, default = 1013.25 mbar is used*/
double attemp,          /* atmospheric temperature in degrees Celsius, only useful with imeth=3 */
double *dgsect,         /* return address for gauquelin sector position */
char *serr);            /* return address for error message */
```

16.6. Auxiliary functions

Coordinate transformation, from ecliptic to equator or vice-versa

```
equator -> ecliptic       : eps must be positive
ecliptic -> equator       : eps must be negative eps, longitude and latitude are in degrees! */

void swe_cotrans(
double *xpo,              /* 3 doubles: long., lat., dist. to be converted; distance remains unchanged,
                             can be set to 1.00 */
      double *xpn,        /* 3 doubles: long., lat., dist. Result of the conversion */
      double eps);        /* obliquity of ecliptic, in degrees. */
```

Coordinate transformation of position and speed, from ecliptic to equator or vice-versa

```
/ * equator -> ecliptic    : eps must be positive
  ecliptic -> equator      : eps must be negative
  eps, long., lat., and speeds in long. and lat. are in degrees! */
void swe_cotrans_sp(
      double *xpo,        /* 6 doubles, input: long., lat., dist. and speeds in long., lat and dist. */
      double *xpn,        /* 6 doubles, position and speed in new coordinate system */
      double eps);        /* obliquity of ecliptic, in degrees. */
```

Get the name of a planet

```
char* swe_get_planet_name(
int ipl,                                  /* planet number */
char* plan_name);       /* address for planet name, at least 20 char */

/* normalization of any degree number to the range 0 ... 360 */
double swe_degnorm(double x);
```

16.7. Other functions that may be useful

PLACALC, the predecessor of SWISSEPH, had included several functions that we do not need for SWISSEPH anymore. Nevertheless we include them again in our DLL, because some users of our software may have taken them over and use them in their applications. However, we gave them new names that were more consistent with SWISSEPH.

PLACALC used angular measurements in centiseconds a lot; a centisecond is **1**/**100** of an arc second. The C type CSEC or centisec is a 32-bit integer. CSEC was used because calculation with integer variables was considerably faster than floating point calculation on most CPUs in 1988, when PLACALC was written.
In the Swiss Ephemeris we have dropped the use of centiseconds and use double (64-bit floating point) for all angular measurements.

Normalize argument into interval [0..DEG360]

```
/ * former function name: csnorm() */
extern EXP32 centisec FAR PASCAL_CONV EXP16 swe_csnorm(centisec p);
```

Distance in centisecs p1 - p2 normalized to [0..360]

```
/ * former function name: difcsn() */
extern EXP32 centisec FAR PASCAL_CONV EXP16 swe_difcsn(centisec p1, centisec p2);
```

Distance in degrees

```
/* former function name: difdegn() */
extern EXP32 double FAR PASCAL_CONV EXP16 swe_difdegn (double p1, double p2);
```

Distance in centisecs p1 - p2 normalized to [-180..180]

```
/* former function name: difcs2n() */
extern EXP32 centisec FAR PASCAL_CONV EXP16 swe_difcs2n(centisec p1, centisec p2);
```

Distance in degrees

```
/* former function name: difdeg2n() */
extern EXP32 double FAR PASCAL_CONV EXP16 swe_difdeg2n(double p1, double p2);
```

Round second, but at 29.5959 always down

```
 /* former function name: roundsec() */
extern EXP32 centisec FAR PASCAL_CONV EXP16 swe_csroundsec(centisec x);
```

Double to long with rounding, no overflow check

```
/* former function name: d2l() */
extern EXP32 long FAR PASCAL_CONV EXP16 swe_d2l(double x);
```

Day of week

```
/*Monday = 0, ... Sunday = 6  former function name: day_of_week() */
extern EXP32 int FAR PASCAL_CONV EXP16 swe_day_of_week(double jd);
```

Centiseconds -> time string

```
/* former function name: TimeString() */
extern EXP32 char *FAR PASCAL_CONV EXP16 swe_cs2timestr(CSEC t, int sep, AS_BOOL
        suppressZero, char *a);
```

Centiseconds -> longitude or latitude string

```
/* former function name: LonLatString() */
extern EXP32 char *FAR PASCAL_CONV EXP16 swe_cs2lonlatstr(CSEC t, char pchar, char mchar,
        char *s);
```

Centiseconds -> degrees string

```
/* former function name: DegreeString() */
extern EXP32 char *FAR PASCAL_CONV EXP16 swe_cs2degstr(CSEC t, char *a);
```

17. The SWISSEPH DLLs

There is a 32 bit DLL: swedll32.dll

You can use our programs swetest.c and swewin.c as examples.To compile swetest or swewin with a DLL:

1. The compiler needs the following files:
swetest.c or swewin.c
swedll32.dll
swedll32.lib (if you choose implicit linking)
swephexp.h
swedll.h
sweodef.h

2. Define the following macros (-d):
   USE_DLL
3. Build swetest.exe from swetest.c and swedll32.lib.
   Build swewin.exe from swewin.c, swewin.rc, and swedll32.lib

We provide some project files which we have used to build our test samples. You will need to adjust the project files to your environment.

We have worked with Microsoft Visual C++ 5.0 (32-bit). The DLLs where built with the Microsoft compilers.

17.1 DLL Interface for brain damaged compilers

If you work with GFA-Basic or some other brain damaged language, the problem will occur that the DLL interface does not support 8-bit, 32-bit, double by value and VOID data or function types. Therefore, we have written a set of modified functions that use double pointers instead of doubles, character pointers instead of characters, and

integers instead of void. The names of these modified functions are the same as the names of their prototypes, except that they end with "_d", e.g. swe_calc_d() instead of swe_calc().The export definitions of these functions can be found in file swedll.h. We do not repeat them here to avoid confusion with the ordinary functions described in the preceding chapters. The additional functions are only wrapper functions, i.e. they call internally the real DLL functions and return the same results.

Swiss Ephemeris release 1.61 is the last release for which 16-bit compilers have been supported and for which a 16-bit DLL has been created.

18. Using the DLL with Visual Basic 5.0

The 32-bit DLL contains the exported function under 'decorated names'. Each function has an underscore before its name, and a suffix of the form @xx where xx is the number of stack bytes used by the call.

The Visual Basic declarations for the DLL functions and for some important flag parameters are in the file \sweph\vb\swedecl.txt and can be inserted directly into a VB program.

A sample VB program vbsweph is included on the distribution, in directory \sweph\vb. To run this sample, the DLL file swedll32.dll must be copied into the vb directory or installed in the Windows system directory.

DLL functions returning a string:
Some DLL functions return a string, e.g.

```
char* swe_get_planet_name(int ipl, char *plname)
```

This function copies its result into the string pointer plname; the calling program must provide sufficient space so that the result string fits into it. As usual in C programming, the function copies the return string into the provided area and returns the pointer to this area as the function value. This allows to use this function directly in a C print statement.

In VB there are three problems with this type of function:

1. The string parameter plname must be initialized to a string of sufficient length before the call; the content does not matter because it is overwritten by the called function. The parameter type must be
   ByVal plname as String.

2. The returned string is terminated by a NULL character. This must be searched in VB and the VB string length must be set accordingly. Our sample program demonstrates how this can be done:

```
Private Function set_strlen(c$) As String
  i = InStr(c$, Chr$(0))
  c$ = Left(c$, i - 1)
  set_strlen = c$
End Function
plname = String(20,0)     ' initialize string to length 20
swe_get_planet_name(SE_SUN, plname)
plname = set_strlen(plname)
```

3. The function value itself is a pointer to character. This function value cannot be used in VB because VB does not have a pointer data type. In VB, such a Function can be either declared as type "As long" and the return value ignored, or it can be declared as a Sub. We have chosen to declare all such functions as ‚Sub', which automatically ignores the return value.
   Declare Sub **swe_get_planet_name** (ByVal ipl as Long, ByVal plname as String)

19. Using the DLL with Borland Delphi and C++ Builder

19.1 Delphi 2.0 and higher (32-bit)

The information in this section was contributed by Markus Fabian, Bern, Switzerland.

In Delphi 2.0 the declaration of the function swe_calc() looks like this:

```
xx : Array[0..5] of double;
function swe_calc (tjd  : double;    // Julian day number
                   ipl  : Integer;   // planet number
                   iflag : Longint;  // flag bits
                   var xx[0] : double;
                   sErr : PChar      // Error-String;
  ) : Longint; stdcall; far; external 'swedll32.dll' Name '_swe_calc@24';
```

A nearly complete set of declarations is in file \sweph\delphi2\swe_d32.pas.
A small sample project for Delphi 2.0 is also included in the same directory (starting with release **1.25** from June 1998). This sample requires the DLL to exist in the same directory as the sample.

19.2 Borland C++ Builder

Borland C++ Builder (BCB) does not understand the Microsoft format in the library file SWEDLL32.LIB; it reports an OMF error when this file is used in a BCB project. The user must create his/her own LIB file for BCB with the utility IMPLIB which is part of BCB.

With the following command command you create a special lib file in the current directory:

```
IMPLIB –f –c swe32bor.lib \sweph\bin\swedll32.dll
```

In the C++ Builder project the following settings must be made:

- Menu Options->Projects->Directories/Conditionals: add the conditional define USE_DLL
- Menu Project->Add_to_project: add the library file swe32bor.lib to your project.
- In the project source, add the include file "swephexp.h"

In the header file swedll.h the declaration for Dllimport must be

```
#define DllImport  extern "C" __declspec( dllimport )
```

This is provided automatically by the __cplusplus switch for release **1.24** and higher. For earlier releases the change must be made manually.

Changes for c++builder 2010
Swiss Ephemeris developer Jean Cremers, Netherlands, reported in August 2010 that a different procedure is needed for new versions of c++builder. This is his report:

> The method described in the file swephprg.htm 'Using the DLL with Borland Delphi and C++ Builder' does not work for the current c++builder (I use c++ builder 6).
> The command 'IMPLIB –f –c swe32bor.lib \sweph\bin\swedll32.dll' will give 2 errors 'unable to open file' and will create the file '-f.lib'.
> It seems implib cannot handle the switches anymore, indeed when left out the file swe32bor.lib is generated but the file is in the coff format.
> A solution is to use the coff2omf.exe utility that comes with c++builder like this 'coff2omf.exe swedll32.dll swe32bor.lib', it will generate a usable lib for c++builder.
>
> The steps to use sweph with c++builder then become:
>
> ---
> With the following command command you create a special lib file in the current directory:
> coff2omf.exe swedll32.dll \sweph\bin\swe32bor.lib
>
> In the C++ Builder project the following settings must be made:
>
> Menu Options->Projects->Directories/Conditionals: add the conditional define USE_DLL
> Menu Project->Add_to_project: add the library file swe32bor.lib to your project.
> In the project source, add the include file "swephexp.h"
> ---

20. Using the Swiss Ephemeris with Perl

The Swiss Ephemeris can be run from Perl using the Perl module SwissEph.pm. The module SwissEph.pm uses XSUB (“eXternal SUBroutine”), which calls the Swiss Ephemeris functions either from a C library or a DLL.

In order to run the Swiss Ephemeris from Perl, you have to

1. Install the Swiss Ephemeris. Either you download the Swiss Ephemeris DLL from http://www.astro.com/swisseph or you download the Swiss Ephemeris C source code and compile a static or dynamic shared library. We built the package on a Linux system and use a shared library of the Swiss Ephemeris functions.

2. Install the XS library:
   - Unpack the file PerlSwissEph-1.76.00.tar.gz (or whatever newest version there is)
   - Open the file Makefile.PL, and edit it according to your requirements. Then run it.
   - make install

If you work on a Windows machine and prefer to use the Swiss Ephemeris DLL, you may want to study Rüdiger Plantiko's Perl module for the Swiss Ephemeris at http://www.astrotexte.ch/sources/SwissEph.zip. There is also a documentation in German language by Rüdiger Plantiko at http://www.astrotexte.ch/sources/swe_perl.html).

21. The C sample program

The distribution contains executables and C source code of sample programs which demonstrate the use of the Swiss Ephemeris DLL and its functions.

All samples programs are compiled with the Microsoft Visual C++ 5.0 compiler (32-bit). Project and Workspace files for these environments are included with the source files.

Directory structure:

Sweph\bin	DLL, LIB and EXE file
Sweph\src	source files, resource files
Sweph\src\swewin32	32-bit windows sample program
Sweph\src\swete32	32-bit character mode sample program

You can run the samples in the following environments:

Swetest.exe	in Windows command line
Swete32.exe	in Windows command line
Swewin32.exe	in Windows

Character mode executable that needs a DLL
Swete32.exe
The project files are in \sweph\src\swete32
swetest.c
swedll32.lib
swephexp.h
swedll.h
sweodef.h
define macros: USE_DLL DOS32 DOS_DEGREE

swewin32.exe
The project files are in \sweph\src\swewin32
swewin.c
swedll32.lib
swewin.rc
swewin.h
swephexp.h
swedll.h
sweodef.h
resource.h
define macro USE_DLL

How the sample programs search for the ephemeris files:

1. check environment variable SE_EPHE_PATH; if it exists it is used, and if it has invalid content, the program fails.

2. Try to find the ephemeris files in the current working directory
3. Try to find the ephemeris files in the directory where the executable resides
4. Try to find a directory named \SWEPH\EPHE in one of the following three drives:
   - where the executable resides
   - current drive
   - drive C:

As soon as it succeeds in finding the first ephemeris file it looks for, it expects all required ephemeris files to reside there. This is a feature of the sample programs only, as you can see in our C code.

The DLL itself has a different and simpler mechanism to search for ephemeris files, which is described with the function swe_set_ephe_path() above.

21. The source code distribution

Starting with release **1.26**, the full source code for the Swiss Ephemeris DLL is made available. Users can choose to link the Swiss Ephemeris code directly into their applications. The source code is written in Ansi C and consists of these files:

Bytes	Date	File name	Comment
1639	Nov 28 17:09	Makefile	unix makefile for library
API interface files			
15050	Nov 27 10:56	swephexp.h	SwissEph API include file
14803	Nov 27 10:59	swepcalc.h	Placalc API include file
Internal files			
8518	Nov 27 10:06	swedate.c	
2673	Nov 27 10:03	swedate.h	
8808	Nov 28 19:24	swedll.h	
24634	Nov 27 10:07	swehouse.c	
2659	Nov 27 10:05	swehouse.h	
31279	Nov 27 10:07	swejpl.c	
3444	Nov 27 10:05	swejpl.h	
38238	Nov 27 10:07	swemmoon.c	
2772	Nov 27 10:05	swemosh.h	
18687	Nov 27 10:07	swemplan.c	
311564	Nov 27 10:07	swemptab.c	
7291	Nov 27 10:06	sweodef.h	
28680	Nov 27 10:07	swepcalc.c	
173758	Nov 27 10:07	sweph.c	
12136	Nov 27 10:06	sweph.h	
55063	Nov 27 10:07	swephlib.c	
4886	Nov 27 10:06	swephlib.h	
43421	Nov 28 19:33	swetest.c	

In most cases the user will compile a linkable or shared library from the source code, using his favorite C compiler, and then link this library with his application.
If the user programs in C, he will only need to include the header file swephexp.h with his application; this in turn will include sweodef.h. All other source files can ignored from the perspective of application development.

22. The PLACALC compatibility API

To simplify porting of older Placalc applications to the Swiss Ephemeris API, we have created the Placalc compatibility API which consists of the header file swepcalc.h. This header file replaces the headers ourdef.h, placalc.h, housasp.h and astrolib.h in Placalc applications.You should be able to link your Placalc aplication now with the Swiss Ephemeris library. The Placalc API is not contained in the SwissEph DLL.
All new software should use the SwissEph API directly.

23. Documentation files

The following files are in the directory \sweph\doc

sweph.cdr	
sweph.gif	
swephin.cdr	
swephin.gif	
swephprg.doc	Documentation for programming, a MS Word-97 file
swephprg.rtf	
swisseph.doc	General information on Swiss Ephemeris
swisseph.rtf	

The files with suffix .CDR are Corel Draw 7.0 documents with the Swiss Ephemeris icons.

24. Swisseph with different hardware and compilers

Depending on what hardware and compiler you use, there will be slight differences in your planetary calculations. For positions in longitude, they will be never larger than **0.0001"** in longitude. Speeds show no difference larger than **0.0002 arcsec/day.**

The following factors show larger differences between HPUX and Linux on a Pentium II processor:
Mean Node, Mean Apogee:
HPUX PA-Risc non-optimized versus optimized code:
differences are smaller than 0.001 arcsec/day

HPUX PA-Risc versus Intel Pentium gcc non-optimzed
differences are smaller than 0.001 arcsec/day

Intel Pentium gss non-optimzed versus -O9 optimized:
Mean Node, True node, Mean Apogee: difference smaller than 0.001 arcsec/day
Osculating Apogee: differences smaller than 0.03 arcsec

The differences originate from the fact that the floating point arithmetic in the Pentium is executed with 80 bit precision, whereas stored program variables have only 64 bit precision. When code is optimized, more intermediate results are kept inside the processor registers, i.e. they are not shortened from 80bit to 64 bit. When these results are used for the next calculation, the outcome is then slightly different.
In the computation of speed for the nodes and apogee, differences between positions at close intervals are involved; the subtraction of nearly equal values results shows differences in internal precision more easily than other types of calculations. As these differences have no effect on any imaginable application software and are mostly within the design limit of Swiss Ephemeris, they can be savely ignored.

25. Debugging and Tracing Swisseph

25.1. If you are using the DLL

Besides the ordinary Swisseph function, there are two additional DLLs that allow you tracing your Swisseph function calls:
Swetrs32.dll is for single task debugging, i.e. if only one application at a time calls Swisseph functions.
Two output files are written:
a) swetrace.txt: reports all Swisseph functions that are being called.
b) swetrace.c: contains C code equivalent to the Swisseph calls that your application did.
The last bracket of the function main() at the end of the file is missing.
If you want to compile the code, you have to add it manually. Note that these files may grow very fast, depending on what you are doing in your application. The output is limited to 10000 function calls per run.
Swetrm32.dll is for multitasking, i.e. if more than one application at a time are calling Swisseph functions. If you used the single task DLL here, all applications would try to write their trace output into the same file.
Swetrm32.dll generates output file names that contain the process identification number of the application by which the DLL is called, e.g. swetrace_192.c and swetrace_192.txt.
Keep in mind that every process creates its own output files and with time might fill your disk.
In order to use a trace DLL, you have to replace your Swisseph DLL by it:

a) save your Swisseph DLL
b) rename the trace DLL as your Swisseph DLL (e.g. as swedll32.dll)

IMPORTANT: The Swisseph DLL will not work properly if you call it from **more than one thread.**

Output samples swetrace.txt:

```
swe_deltat: 2451337.870000    0.000757
swe_set_ephe_path: path_in =  path_set = \sweph\ephe\
swe_calc: 2451337.870757      -1    258   23.437404   23.439365  -0.003530  -0.001961  0.000000  0.000000
swe_deltat: 2451337.870000    0.000757
swe_sidtime0: 2451337.870000 sidt = 1.966683          eps = 23.437404        nut = -0.003530
swe_sidtime: 2451337.870000   1.966683
swe_calc: 2451337.870757      0     258   77.142261   -0.000071  1.014989   0.956743   -0.000022 0.000132
swe_get_planet_name: 0        Sun
```

swetrace.c:

```
#include "sweodef.h"
#include "swephexp.h"

void main()
{
  double tjd, t, nut, eps; int i, ipl, retc; long iflag;
  double armc, geolat, cusp[12], ascmc[10]; int hsys;
  double xx[6]; long iflgret;
  char s[AS_MAXCH], star[AS_MAXCH], serr[AS_MAXCH];

/*SWE_DELTAT*/
  tjd = 2451337.870000000; t = swe_deltat(tjd);
  printf("swe_deltat: %f\t%f\t\n", tjd, t);

/*SWE_CALC*/
  tjd = 2451337.870757482; ipl = 0; iflag = 258;
  iflgret = swe_calc(tjd, ipl, iflag, xx, serr);      /* xx = 1239992 */

/*SWE_CLOSE*/
  swe_close();
```

25.2 If you are using the source code

Similar tracing is also possible if you compile the Swisseph source code into your application. Use the preprocessor definitions TRACE=1 for single task debugging, and TRACE=2 for multitasking. In most compilers this flag can be set with -DTRACE=1 or /DTRACE=1.
For further explanations, see 21.1.

Appendix

Update and release history

Updated	By	
30-sep-97	Alois	added chapter 10 (sample programs)
7-oct-97	Dieter	inserted chapter 7 (house calculation)
8-oct-97	Dieter	Appendix ”Changes from version 1.00 to 1.01”
12-oct-1997	Alois	Added new chapter 10 Using the DLL with Visual Basic
26-oct-1997	Alois	improved implementation and documentation of swe_fixstar()
28-oct-1997	Dieter	Changes from Version 1.02 to 1.03
29-oct-1997	Alois	added VB sample extension, fixed VB declaration errors
9-Nov-1997	Alois	added Delphi declaration sample
8-Dec-97	Dieter	remarks concerning computation of asteroids, changes to version 1.04
8-Jan-98	Dieter	changes from version 1.04 to 1.10.
12-Jan-98	Dieter	changes from version 1.10 to 1.11.
21-Jan-98	Dieter	calculation of topocentric planets and house positions (1.20)

28-Jan-98	Dieter	Delphi 1.0 sample and declarations for 16- and 32-bit Delphi (1.21)
11-Feb-98	Dieter	version 1.23
7-Mar-1998	Alois	version 1.24 support for Borland C++ Builder added
4-June-1998	Alois	version 1.25 sample for Borland Delphi-2 added
29-Nov-1998	Alois	version 1.26 source code information added §16, Placalc API added
1-Dec-1998	Dieter	chapter 19 and some additions in beginning of Appendix.
2-Dec-1998	Alois	Equation of Time explained (in §4), changes version 1.27 explained
3-Dec-1998	Dieter	Note on ephemerides of 1992 QB1 and 1996 TL66
17-Dec-1998	Alois	Note on extended time range of 10'800 years
22 Dec 1998	Alois	Appendix A
12-Jan-1999	Dieter	Eclipse functions added, version 1.31
19-Apr-99	Dieter	version 1.4
8-Jun-99	Dieter	Chapter 21 on tracing an debugging Swisseph
27-Jul-99	Dieter	Info about sidereal calculations
16-Aug-99	Dieter	version 1.51, minor bug fixes
15-Feb-00	Dieter	many things for version 1.60
19-Mar-00	Vic Ogi	SWEPHPRG.DOC re-edited
17-apr-02	Dieter	Documentation for version 1.64
26-Jun-02	Dieter	Version 1.64.01
31-dec-2002	Alois	edited doc to remove references to 16-bit version
12-jun-2003	Alois/Dieter	Documentation for version 1.65
10-Jul-2003	Dieter	Documentation for version 1.66
25-May-2004	Dieter	Documentation of eclipse functions updated
31-Mar-2005	Dieter	Documentation for version 1.67
3-May-2005	Dieter	Documentation for version 1.67.01
22-Feb-2006	Dieter	Documentation for version 1.70.00
2-May-2006	Dieter	Documentation for version 1.70.01
5-Feb-2006	Dieter	Documentation for version 1.70.02
30-Jun-2006	Dieter	Documentation for version 1.70.03
28-Sep-2006	Dieter	Documentation for version 1.71
29-May-2008	Dieter	Documentation for version 1.73
18-Jun-2008	Dieter	Documentation for version 1.74
27-Aug-2008	Dieter	Documentation for version 1.75
7-April-2009	Dieter	Documentation of version 1.76

Release	**Date**	
1.00	30-sep-1997	
1.01	9-oct-1997	houses(), sidtime() made more convenient for developer, Vertex added.
1.02	16-oct-1997	houses() changed again, Visual Basic support, new numbers for fictitious planets This release was pushed to all existing licensees at this date.
1.03	28-Oct-1997	minor bug fixes, improved swe_fixstar() functionality. This release was not pushed, as the changes and bug fixes are minor; no changes of function definitions occurred.
1.04	8-Dec-1997	minor bug fixes; more asteroids.
1.10	9-Jan-1998	bug fix, s. Appendix. This release was pushed to all existing licensees at this date.
1.11	12-Jan-98	small improvements
1.20	20-Jan-98	New: topocentric planets and house positions; a minor bug fix
1.21	28-Jan-98	Delphi declarations and sample for Delphi 1.0
1.22	2-Feb-98	Asteroids moved to subdirectory. Swe_calc() finds them there.
1.23	11-Feb-98	two minor bug fixes.
1.24	7-Mar-1998	Documentation for Borland C++ Builder added, see section 14.3
1.25	4-June-1998	Sample for Borland Delphi-2 added
1.26	29-Nov-1998	full source code made available, Placalc API documented
1.27	2-dec-1998	Changes to SE_EPHE_PATH and swe_set_ephe_path()
1.30	17-Dec-1998	Time range extended to 10'800 years
1.31	12-Jan-1999	New: Eclipse functions added
1.40	19-Apr-99	New: planetary phenomena added; bug fix in swe_sol_ecl_when_glob();
1.50	27-Jul-99	New: SIDEREAL planetary positions and houses; new fixstars.cat
1.51	16-Aug-99	Minor bug fixes
1.60	15-Feb-2000	Major release with many new features and some minor bug fixes
1.61	11-Sep-2000	Minor release, additions to se_rise_trans(), swe_houses(), ficitious planets
1.61.01	18-Sep-2000	Minor release, added Alcabitus house system
1.61.02	10-Jul-2001	Minor release, fixed bug which prevented asteroid files > 22767 to be accepted
1.61.03	20-Jul-2001	Minor release, fixed bug which was introduced in 1.61.02: Ecliptic was computed in Radians instead of degrees

1.62.00	23-Jul-2001	Minor release, several bug fixes, code for fictitious satellites of the earth, asteroid files > 55535 are accepted
1.62.01	16-Oct-2001	Bug fix, string overflow in sweph.c::read_const(),
1.63.00	5-Jan-2002	Added house calculation to sweetest.c and swetest.exe
1.64.00	6-Mar-2002	House system 'G' for house functions and function swe_gauquelin_sector() for Gauquelin sector calculations Occultations of planets and fixed stars by the moon New Delta T algorithms
1.64.01	26-Jun-2002	Bug fix in swe_fixstar(). Stars with decl. between –1° and 0° were wrong
1.65.00	12-Jun-2003	Long variables replaced by INT32 for 64-bit compilers
1.66.00	10-Jul-2003	House system 'M' for Morinus houses
1.67.00	31-Mar-2005	Update Delta T
1.67.01	3-May-2005	Docu for sidereal calculations (Chap. 10) updated (precession-corrected transits)
1.70.00	22-Feb-2006	all relevant IAU resolutions up to 2005 have been implemented
1.70.01	2-May-2006	minor bug fix
1.70.02	5-May-2006	minor bug fix
1.70.03	30-June-2006	bug fix
1.71	28-Sep-2006	Swiss Ephemeris functions able to calculate minor planet no 134340 Pluto
1.72	28-Sep-2007	New function swe_refract_extended(), Delta T update, minor bug fixes
1.73	29-May-2008	New function swe_fixstars_mag(), Whole Sign houses
1.74	18-Jun-2008	Bug fixes
1.75	27-Aug-2008	Swiss Ephemeris can read newer JPL ephemeris files; bug fixes
1.76	7-April-2009	Heliacal risings, UTC and minor improvements/bug fixes
1.77	26-Jan-2010	swe_deltat(), swe_fixstar() improved, swe_utc_time_zone_added
1.78	3-Aug-2012	New precession, improvement of some eclipse functions, some minor bug fixes
1.79	18-Apr-2013	New precession, improvement of some eclipse functions, some minor bug fixes

Changes from version 1.78 to 1.79

- Improved precision in eclipse calculations: 2nd and 3rd contact with solar eclipses, penumbral and partial phases with lunar eclipses.
- Bug fix in function swe_sol_eclipse_when_loc().If the local maximum eclipse occurs at sunset or sunrise, tret[0] now gives the moment when the lower limb of the Sun touches the horizon. This was not correctly implemented in former versions
- Several changes to C code that had caused compiler warnings (as proposed by Torsten Förtsch).
- Bug fix in Perl functions swe_house() etc. These functions had crashed with a segmention violation if called with the house parameter 'G'.
- Bug fix in Perl function swe_utc_to_jd(), where gregflag had been read from the 4th instead of the 6th parameter.
- Bug fix in Perl functions to do with date conversion. The default mechanism for gregflag was buggy.
- For Hindu astrologers, some more ayanamshas were added that are related to Suryasiddhanta and Aryabhata and are of historical interest.

Changes from version 1.77 to 1.78

- precession is now calculated according to Vondrák, Capitaine, and Wallace 2011.
- Delta t for current years updated.
- new function: swe_rise_trans_true_hor() for risings and settings at a local horizon with known height.
- functions swe_sol_eclipse_when_loc(), swe_lun_occult_when_loc(): return values tret[5] and tret[6] (sunrise and sunset times) added, which had been 0 so far.
- function swe_lun_eclipse_how(): return values attr[4-6] added (azimuth and apparent and true altitude of moon).
- Attention with swe_sol_eclipse_how(): return value attr[4] is azimuth, now measured from south, in agreement with the function swe_azalt() and swe_azalt_rev().
- minor bug fix in swe_rise_trans(): twilight calculation returned invalid times at high geographic latitudes.
- minor bug fix: when calling swe_calc() 1. with SEFLG_MOSEPH, 2. with SEFLG_SWIEPH, 3. again with SEFLG_MOSEPH, the result of 1. and 3. were slightly different. Now they agree.
- minor bug fix in swe_houses(): With house methods H (Horizon), X (Meridian), M (Morinus), and geographic latitudes beyond the polar circle, the ascendant was wrong at times. The ascendant always has to be on the eastern part of the horizon.

Changes from version 1.76 to 1.77

- Delta T:
  - Current values were updated.
  - File sedeltat.txt understands doubles.

- For the period before 1633, the new formulae by Espenak and Meeus (2006) are used. These formulae were derived from Morrison & Stephenson (2004), as used by the Swiss Ephemeris until version 1.76.02.
- The tidal acceleration of the moon contained in LE405/6 was corrected according to Chapront/Chapront-Touzé/Francou A&A 387 (2002), p. 705.

- Fixed stars:
- There was an error in the handling of the proper motion in RA. The values given in fixstars.cat, which are taken from the Simbad database (Hipparcos), are referred to a great circle and include a factor of cos(d0).
- There is a new fixed stars file sefstars.txt. The parameters are now identical to those in the Simbad database, which makes it much easier to add new star data to the file. If the program function swe_fixstars() does not find sefstars.txt, it will try the the old fixed stars file fixstars.cat and will handle it correctly.
- Fixed stars data were updated, some errors corrected.
- Search string for a star ignores white spaces.

- Other changes:
- New function swe_utc_time_zone(), converts local time to UTC and UTC to local time. Note, the function has no knowledge about time zones. The Swiss Ephemeris still does not provide the time zone for a given place and time.
- swecl.c:swe_rise_trans() has two new minor features: SE_BIT_FIXED_DISC_SIZE and SE_BIT_DISC_BOTTOM (thanks to Olivier Beltrami)
- minor bug fix in swemmoon.c, Moshier's lunar ephemeris (thanks to Bhanu Pinnamaneni)
- solar and lunar eclipse functions provide additional data:
  attr[8] magnitude, attr[9] saros series number, attr[10] saros series member number

Changes from version 1.75 to 1.76

New features:
- Functions for the calculation of heliacal risings and related phenomena, s. chap. 6.15-6.17.
- Functions for conversion between UTC and JD (TT/UT1), s. chap. 7.2 and 7.3.
- File sedeltat.txt allows the user to update Delta T himself regularly, s. chap. 8.3
- Function swe_rise_trans(): twilight calculations (civil, nautical, and astronomical) added
- Function swe_version() returns version number of Swiss Ephemeris.
- Swiss Ephemeris for Perl programmers using XSUB

Other updates:
- Delta T updated (-2009).

Minor bug fixes:
- swe_house_pos(): minor bug with Alcabitius houses fixed
- swe_sol_eclipse_when_glob(): totality times for eclipses jd2456776 and jd2879654 fixed (tret[4], tret[5])

Changes from version 1.74 to version 1.75

- The Swiss Ephemeris is now able to read ephemeris files of JPL ephemerides DE200 - DE421. If JPL will not change the file structure in future releases, the Swiss Ephemeris will be able to read them, as well.

- Function swe_fixstar() (and swe_fixstar_ut()) was made slightly more efficient.

- Function swe_gauquelin_sector() was extended.

- Minor bug fixes.

Changes from version 1.73 to version 1.74

The Swiss Ephemeris is made available under a dual licensing system:
a) GNU public license version 2 or later
b) Swiss Ephemeris Professional License
For more details, see at the beginning of this file and at the beginning of every source code file.

Minor bug fixes:
- Bug in swe_fixstars_mag() fixed.
- Bug in swe_nod_aps() fixed. With retrograde asteroids (20461 Dioretsa, 65407 2002RP120), the calculation of perihelion and aphelion was not correct.
- The ephemeris of asteroid 65407 2002RP120 was updated. It had been wrong before 17 June 2008.

Changes from version 1.72 to version 1.73

New features:
- Whole Sign houses implemented (W)
- swe_house_pos() now also handles Alcabitius house method
- function swe_fixstars_mag() provides fixed stars magnitudes

Changes from version 1.71 to version 1.72

- Delta T values for recent years were updated
- Delta T calculation before 1600 was updated to Morrison/Stephenson 2004..
- New function swe_refract_extended(), in cooperation with archaeoastronomer Victor Reijs.
  This function allows correct calculation of refraction for altitudes above sea > 0, where the ideal horizon and Planets that are visible may have a negative height.
- Minor bugs in swe_lun_occult_when_glob() and swe_lun_eclipse_how() were fixed.

Changes from version 1.70.03 to version 1.71

In September 2006, Pluto was introduced to the minor planet catalogue and given the catalogue number 134340. The numerical integrator we use to generate minor planet ephemerides would crash with 134340 Pluto, because Pluto is one of those planets whose gravitational perturbations are used for the numerical integration. Instead of fixing the numerical integrator for this special case, we chang the Swiss Ephemeris functions in such a way that they treat minor planet 134340 Pluto (ipl=SE_AST_OFFSET+134340) as our main body Pluto (ipl=SE_PLUTO=9). This also results in a slightly better precision for 134340 Pluto.

Swiss Ephemeris versions prior to 1.71 are not able to do any calculations for minor planet number 134340.

Changes from version 1.70.02 to version 1.70.03

Bug fixed (in swecl.c: swi_bias()): This bug sometimes resulted in a crash, if the DLL was used and the SEFLG_SPEED was not set. It seems that the error happened only with the DLL and did not appear, when the Swiss Ephemeris C code was directly linked to the application.

Code to do with (#define NO_MOSHIER) war removed.

Changes from version 1.70.01 to version 1.70.02

Bug fixed in speed calculation for interpolated lunar apsides. With ephemeris positions close to 0 Aries, speed calculations were completely wrong. E.g. swetest -pc -bj3670817.276275689 (speed = 1448042° !)
Thanks, once more, to Thomas Mack, for testing the software so well.

Changes from version 1.70.00 to version 1.70.01

Bug fixed in speed calculation for interpolated lunar apsides. Bug could result in program crashes if the speed flag was set.

Changes from version 1.67 to version 1.70

Update of algorithms to IAU standard recommendations:
All relevant IAU resolutions up to 2005 have been implemented. These include:
- the "frame bias" rotation from the JPL reference system ICRS to J2000. The correction of position ~= 0.0068 arc sec in right ascension.
- the precession model P03 (Capitaine/Wallace/Chapront 2003). The correction in longitude is smaller than 1 arc second from 1000 B.C. on.
- the nutation model IAU2000B (can be switched to IAU2000A)
- corrections to epsilon
- corrections to sidereal time
- fixed stars input data can be "J2000" or "ICRS"
- fixed stars conversion FK5 -> J2000, where required
- fixed stars data file was updated with newer data
- constants in sweph.h updated

For more info, see the documentation swisseph.doc, chapters 2.1.2.1-3.

New features:

- Ephemerides of "interpolated lunar apogee and perigee", as published by Dieter Koch in 2000 (swetest -pcg).
  For more info, see the documentation swisseph.doc, chapter 2.2.4.
- House system according to Bogdan Krusinski (character 'U').
  For more info, see the documentation swisseph.doc, chapter 6.1.13.

Bug fixes:
- Calculation of magnitude was wrong with asteroid numbers < 10000 (10-nov-05)

Changes from version 1.66 to version 1.67

Delta-T updated with new measured values for the years 2003 and 2004, and better estimates for 2005 and 2006.
Bug fixed #define SE_NFICT_ELEM 15

Changes from version 1.65 to version 1.66

New features:
House system according to Morinus (system 'M').

Changes from version 1.64.01 to version 1.65.00

'long' variables were changed to 'INT32' for 64-bit compilers.

Changes from version 1.64 to version 1.64.01

- Bug fixed in swe_fixstar(). Declinations between –1° and 0° were wrongly taken as positive.
Thanks to John Smith, Serbia, who found this bug.
- Several minor bug fixes and cosmetic code improvements suggested by Thomas Mack, Germany.
  swetest.c: options –po and –pn work now.
  Sweph.c: speed of mean node and mean lunar apogee were wrong in rare cases, near 0 Aries.

Changes from version 1.63 to version 1.64

New features:
1) Gauquelin sectors:
- swe_houses() etc. can be called with house system character 'G' to calculate Gauquelin sector boundaries.
- swe_house_pos() can be called with house system 'G' to calculate sector positions of planets.
- swe_gauquelin_sector() is new and calculates Gauquelin sector positions with three methods: without ecl. latitude, with ecl. latitude, from rising and setting.

2) Waldemath Black Moon elements have been added in seorbel.txt (with thanks to Graham Dawson).

3) Occultations of the planets and fixed stars by the moon
- swe_lun_occult_when_loc() calculates occultations for a given geographic location
- swe_lun_occult_when_glob() calculates occultations globally

4) Minor bug fixes in swe_fixstar() (Cartesian coordinates), solar eclipse functions, swe_rise_trans()

5) sweclips.c integrated into swetest.c. Swetest now also calculates eclipses, occultations, risings and settings.

6) new Delta T algorithms

Changes from version 1.62 to version 1.63

New features:
The option –house was added to swetest.c so that swetest.exe can now be used to compute complete horoscopes in textual mode.
Bux fix: a minor bug in function swe_co_trans was fixed. It never had an effect.

Changes from version 1.61.03 to version 1.62

New features:
1) Elements for hypothetical bodies that move around the earth (e.g. Selena/White Moon) can be added to the file seorbel.txt.
2) The software will be able to read asteroid files > 55535.

Bug fixes:
1) error in geocentric planetary descending nodes fixed
2) swe_calc() now allows hypothetical planets beyond SE_FICT_OFFSET + 15
3) position of hypothetical planets slightly corrected (< 0.01 arc second)

Changes from version 1.61 to 1.61.01

New features:
1. swe_houses and swe_houses_armc now supports the Alcabitus house system. The function swe_house_pos() does not yet, because we wanted to release quickly on user request.

Changes from version 1.60 to 1.61

New features:
1. Function swe_rise_trans(): Risings and settings also for disc center and without refraction
2. “topocentric” house system added to swe_houses() and other house-related functions
3. Hypothetical planets (seorbel.txt), orbital elements with t terms are possible now (e.g. for Vulcan according to L.H. Weston)

Changes from version 1.51 to 1.60

New features:
1. Universal time functions swe_calc_ut(), swe_fixstar_ut(), etc.
2. Planetary nodes, perihelia, aphelia, focal points
3. Risings, settings, and meridian transits of the Moon, planets, asteroids, and stars.
4. Horizontal coordinates (azimuth and altitude)
5. Refraction
6. User-definable orbital elements
7. Asteroid names can be updated by user
8. Hitherto missing "Personal Sensitive Points" according to M. Munkasey.

Minor bug fixes:

- **Astrometric lunar positions** (not relevant for astrology; swe_calc(tjd, SE_MOON, SEFLG_NOABERR)) had a maximum error of about 20 arc sec).
- **Topocentric lunar positions** (not relevant for common astrology): the ellipsoid shape of the earth was not correctly implemented. This resulted in an error of 2 - 3 arc seconds. The new precision is 0.2 - 0.3 arc seconds, corresponding to about 500 m in geographic location. This is also the precision that Nasa's Horizon system provides for the topocentric moon. The planets are much better, of course.
- **Solar eclipse functions***:* The correction of the topocentric moon and another small bug fix lead to slightly different results of the solar eclipse functions. The improvement is within a few time seconds.

Changes from version 1.50 to 1.51

Minor bug fixes:

- J2000 coordinates for the lunar node and osculating apogee corrected. This bug did not affect ordinary computations like ecliptical or equatorial positions.
- minor bugs in swetest.c corrected
- sweclips.exe recompiled
- trace DLLs recompiled
- some VB5 declarations corrected

Changes from version 1.40 to 1.50

New: SIDEREAL planetary and house position.

- The fixed star file fixstars.cat has been improved and enlarged by Valentin Abramov, Tartu, Estonia.
- Stars have been ordered by constellation. Many names and alternative spellings have been added.
- Minor bug fix in solar eclipse functions, sometimes relevant in border-line cases annular/total, partial/total.
- J2000 coordinates for the lunar nodes were redefined: In versions before 1.50, the J2000 lunar nodes were the intersection points of the lunar orbit with the ecliptic of 2000. From 1.50 on, they are defined as the intersection points with the ecliptic of date, referred to the coordinate system of the ecliptic of J2000.

Changes from version 1.31 to 1.40

New: Function for several planetary phenomena added
Bug fix in swe_sol_ecl_when_glob(). The time for maximum eclipse at local apparent noon (tret[1]) was sometimes wrong. When called from VB5, the program crashed.

Changes from version 1.30 to 1.31

New: Eclipse functions added.
Minor bug fix: with previous versions, the function swe_get_planet_name() got the name wrong, if it was an asteroid name and consisted of two or more words (e.g. Van Gogh)

Changes from version 1.27 to 1.30

The time range of the Swiss Ephemeris has been extended by numerical integration. The Swiss Ephemeris now covers the period **2 Jan 5401 BC** to **31 Dec 5399 AD**. To use the extended time range, the appropriate ephemeris files must be downloaded.
In the JPL mode and the Moshier mode the time range remains unchanged at 3000 BC to 3000 AD.
IMPORTANT
Chiron's ephemeris is now restricted to the time range **650 AD – 4650 AD**; for explanations, see swisseph.doc.
Outside this time range, Swiss Ephemeris returns an error code and a position value 0. You must handle this situation in your application. There is a similar restriction with Pholus (as with some other asteroids).

Changes from version 1.26 to 1.27

The environment variable SE_EPHE_PATH is now always overriding the call to swe_set_ephe_path() if it is set and contains a value.
Both the environment variable and the function argument can now contain a list of directory names where the ephemeris files are looked for. Before this release, they could contain only a single directory name.

Changes from version 1.25 to 1.26

- The asteroid subdirectory ephe/asteroid has been split into directories ast0, ast1,... with 1000 asteroid files per directory.
- source code is included with the distribution under the new licensing model
- the Placalc compatibility API (swepcalc.h) is now documented
- There is a new function to compute the equation of time swe_time_equ().
- Improvements of ephemerides:
- **ATTENTION:** Ephemeris of **16 Psyche** has been wrong so far ! By a mysterious mistake it has been identical to 3 Juno.
- Ephemerides of Ceres, Pallas, Vesta, Juno, Chiron and Pholus have been reintegrated, with more recent orbital elements and parameters (e.g. asteroid masses) that are more appropriate to Bowells database of minor planets elements. The differences are small, though.
- Note that the CHIRON ephemeris is should not be used before **700 A.D.**
- Minor bug fix in computation of topocentric planet positions. Nutation has not been correcly considered in observer's position. This has lead to an error of 1 milliarcsec with the planets and 0.1" with the moon.
- We have inactivated the coordinate transformation from **IERS** to **FK5**, because there is still no generally accepted algorithm. This results in a difference of a few milliarcsec from former releases.

Changes from version 1.22 to 1.23

- The topocentric flag now also works with the fixed stars. (The effect of diurnal aberration is a few 0.1 arc second.)
- Bug fix: The return position of swe_cotrans_sp() has been 0, when the input distance was 0.
- About 140 asteroids are on the CD.

Changes from version 1.21 to 1.22

- Asteroid ephemerides have been moved to the ephe\asteroid.
- The DLL has been modified in such a way that it can find them there.
- All asteroids with catalogue number below 90 are on the CD and a few additional ones.

Changes from version 1.20 to 1.21

Sample program and function declarations for Delphi 1.0 added.

Changes from version 1.11 to 1.20

New:

- A flag bit SEFLG_TOPOCTR allows to compute topocentric planet positions. Before calling swe_calc(), call swe_set_topo.
- swe_house_pos for computation of the house position of a given planet. See description in SWISSEPH.DOC, Chapter 3.1 "Geocentric and topocentric positions". A bug has been fixed that has sometimes turned up, when the JPL ephemeris was closed. (An error in memory allocation and freeing.)
- Bug fix: swe_cotrans() did not work in former versions.

Changes from version 1.10 to 1.11

No bug fix, but two minor improvements:

- A change of the ephemeris bits in parameter **iflag** of function swe_calc() usually forces an implicit swe_close() operation. Inside a loop, e.g. for drawing a graphical epehemeris, this can slow down a program. Before this release, two calls with iflag = 0 and iflag = SEFLG_SWIEPH where considered different, though in fact the same ephemeris is used. Now these two calls are considered identical, and swe_close() is not performed implicitly.
  For calls with the pseudo-planet-number ipl = SE_ECL_NUT, whose result does not depend on the chosen ephemeris, the ephemeris bits are ignored completely and swe_close() is never performed implicitly.
- In former versions, calls of the Moshier ephemeris with speed and without speed flag have returned a very small difference in position (0.01 arc second). The reason was that, for precise speed, swe_calc() had to do an additional iteration in the light-time calculation. The two calls now return identical position data.

Changes from version 1.04 to 1.10

- A bug has been fixed that sometimes occurred in swe_calc() when the user changed **iflag** between calls, e.g. the speed flag. The first call for a planet which had been previously computed for the same time, but a different iflag, could return incorrect results, if Sun, Moon or Earth had been computed for a different time in between these two calls.
- More asteroids have been added in this release.

Changes from Version 1.03 to 1.04

- A bug has been fixed that has sometimes lead to a floating point exception when the speed flag was not specified and an unusual sequence of planets was called.
- Additional asteroid files have been included.

Attention: Use these files only with the new DLL. Previous versions cannot deal with more than one additional asteroid besides the main asteroids. This error did not appear so far, because only 433 Eros was on our CD-ROM.

Changes from Version 1.02 to 1.03

- swe_fixstar() has a better implementation for the search of a specific star. If a number is given, the non-comment lines in the file fixstars.cat are now counted from 1; they where counted from zero in earlier releases.
- swe_fixstar() now also computes heliocentric and barycentric fixed stars positions. Former versions Swiss Ephemeris always returned geocentric positions, even if the heliocentric or the barycentric flag bit was set.
- The Galactic Center has been included in fixstars.cat.
- Two small bugs were fixed in the implementation of the barycentric Sun and planets. Under unusual conditions, e.g. if the caller switched from JPL to Swiss Ephemeris or vice-versa, an error of an arc second appeared with the barycentric sun and 0.001 arc sec with the barycentric planets. However, this did not touch normal geocentric computations.

- Some VB declarations in swedecl.txt contained errors and have been fixed. The VB sample has been extended to show fixed star and house calculation. This fix is only in 1.03 releases from 29-oct-97 or later, not in the two 1.03 CDROMs we burned on 28-oct-97.

Changes from Version 1.01 to 1.02

- The function swe_houses() has been changed.
- A new function swe_houses_armc() has been added which can be used when a sidereal time (**armc**) is given but no actual date is known, e.g. for Composite charts.
- The body numbers of the hypothetical bodies have been changed.
- The development environment for the DLL and the sample programs have been changed from Watcom 10.5 to Microsoft Visual C++ (5.0 and 1.5). This was necessary because the Watcom compiler created LIB files which were not compatible with Microsoft C. The LIB files created by Visual C however are compatible with Watcom.

Changes from Version 1.00 to 1.01

1. Sidereal time

The computation of the sidereal time is now much easier. The obliquity and nutation are now computed inside the function. The structure of the function swe_sidtime() has been changed as follows:

```
/* sidereal time */
double swe_sidtime(double tjd_ut);      /* Julian day number, UT */
```

The old functions swe_sidtime0() has been kept for backward compatibility.

2. Houses

The calculation of houses has been simplified as well. Moreover, the Vertex has been added.
The version **1.01** structure of swe_houses() is:

```
int swe_houses(
        double tjd_ut,      /* julian day number, UT */
        double geolat,      /* geographic latitude, in degrees */
        double geolon,      /* geographic longitude, in degrees */
        char hsys,          /* house method, one of the letters PKRCAV */
        double *asc,        /* address for ascendant */
        double *mc,         /* address for mc */
        double *armc,       /* address for armc */
        double *vertex,     /* address for vertex */
double *cusps);             /* address for 13 doubles: 1 empty + 12 houses */
```

Note also, that the indices of the cusps have changed:

```
cusp[0] = 0             (before: cusp[0] = house 1)
cusp[1] = house 1       (before: cusp[1] = house 2)
cusp[2] = house 2       (etc.)
```

etc.

3. Ecliptic obliquity and nutation

The new pseudo-body SE_ECL_NUT replaces the two separate pseudo-bodies SE_ECLIPTIC and SE_NUTATION in the function swe_calc().

Appendix A

What is missing ?

There are some important limits in regard to what you can expect from an ephemeris module. We do not tell you: how to draw a chart

- which glyphs to use
- when a planet is stationary (it depends on you how slow you want it to be)
- how to compute universal time from local time, i.e. what timezone a place is located in
- how to compute progressions, solar returns, composit charts, transit times and a lot else
- what the different calendars (Julian, Gregorian, ..) mean and when they applied.

Index

Flag

Default ephemeris flag
Ephemeris flags
Flag bits
Speed flag

Body, Point

Additional asteroids
Fictitious planets
Find a name
How to compute
Special body SE_ECL_NUT
Uranian planets

Position

Astrometric
Barycentric
Equatorial
Heliocentric
J2000
Position and Speed
Radians/degrees
Sidereal
Topocentric
True geometrical position
True/apparent
x, y, z

What is..

Ayanamsha
Dynamical Time
Ephemeris Time
Equation of time
Julian day
Universal Time
Vertex/Anivertex

How to…

Change the tidal acceleration
compute sidereal composite house cusps
compute the composite ecliptic obliquity
Draw the eclipse path
Get obliquity and nutation
Get the umbra/penumbra limits
Search for a star
Switch the coordinate systems
Switch true/mean equinox of date

Errors

Asteroids
Avoiding Koch houses
Ephemeris path length
Errors and return values
Fatal error
House cusps beyond the polar circle
Koch houses limitations
Overriding environment variables
Speeds of the fixed stars

Variable

Armc
Ascmc[..]
Atpress
Attemp
Ayan_t0
Cusps[..]
Eps
Gregflag
Hsys
Iflag
Ipl
Method
Rsmi
Sid_mode
Star

Function — Description

Swe_azalt — Computes the horizontal coordinates (azimuth and altitude)
Swe_azalt_rev — computes either ecliptical or equatorial coordinates from azimuth and true altitude
swe_calc — computes the positions of planets, asteroids, lunar nodes and apogees
swe_calc_ut — Modified version of swe_calc
swe_close — releases most resources used by the Swiss Ephemeris
swe_cotrans — Coordinate transformation, from ecliptic to equator or vice-versa
swe_cotrans_sp — Coordinate transformation of position and speed, from ecliptic to equator or vice-versa
swe_date_conversion — computes a Julian day from year, month, day, time and checks whether a date is legal
swe_degnorm — normalization of any degree number to the range 0 ... 360
swe_deltat — Computes the difference between Universal Time (UT, GMT) and Ephemeris time

swe_fixstar	computes fixed stars
swe_fixstar_ut	Modified version of swe_fixstar
swe_get_ayanamsa	Computes the ayanamsha
swe_get_ayanamsa_ut	Modified version of swe_get_ayanamsa
swe_get_planet_name	Finds a planetary or asteroid name by given number
swe_get_tid_acc	Gets the tidal acceleration
swe_house_pos	compute the house position of a given body for a given ARMC
swe_houses	Calculates houses for a given date and geographic position
swe_houses_armc	computes houses from ARMC (e.g. with the composite horoscope which has no date)
swe_houses_ex	the same as swe_houses(). Has a parameter, which can be used, if sidereal house positions are wanted
swe_julday	Conversion from day, month, year, time to Julian date
swe_lun_eclipse_how	Computes the attributes of a lunar eclipse at a given time
swe_lun_eclipse_when	Finds the next lunar eclipse
swe_nod_aps	Computes planetary nodes and apsides: perihelia, aphelia, second focal points of the orbital ellipses
swe_nod_aps_ut	Modified version of swe_nod_aps
swe_pheno	Function computes phase, phase angle, elongation, apparent diameter, apparent magnitude
swe_pheno_ut	Modified version of swe_pheno
swe_refrac	The true/apparent altitude convertion
swe_revjul	Conversion from Julian date to day, month, year, time
swe_rise_trans	Computes the times of rising, setting and meridian transits
swe_set_ephe_path	Set application's own ephemeris path
swe_set_jpl_file	Sets JPL ephemeris directory path
swe_set_sid_mode	Specifies the sidereal modes
swe_set_tid_acc	Sets tidal acceleration used in swe_deltat()
swe_set_topo	Sets what geographic position is to be used before topocentric planet positions for a certain birth place can be computed
swe_sidtime	returns sidereal time on Julian day
swe_sidtime0	returns sidereal time on Julian day, obliquity and nutation
swe_sol_eclipse_how	Calculates the solar eclipse attributes for a given geographic position and time
swe_sol_eclipse_when_glob	finds the next solar eclipse globally
swe_sol_eclipse_when_loc	finds the next solar eclipse for a given geographic position
swe_sol_eclipse_where	finds out the geographic position where an eclipse is central or maximal
swe_time_equ	returns the difference between local apparent and local mean time

PlaCalc function	Description
swe_csnorm	Normalize argument into interval [0..DEG360]
swe_cs2degstr	Centiseconds -> degrees string
swe_cs2lonlatstr	Centiseconds -> longitude or latitude string
swe_cs2timestr	Centiseconds -> time string
swe_csroundsec	Round second, but at 29.5959 always down
swe_d2l	Double to long with rounding, no overflow check
swe_day_of_week	Day of week Monday = 0, ... Sunday = 6
swe_difcs2n	Distance in centisecs p1 – p2 normalized to [-180..180]
swe_difcsn	Distance in centisecs p1 – p2 normalized to [0..360]
swe_difdeg2n	Distance in degrees
swe_difdegn	Distance in degrees

End of SWEPHPRG.DOC