

5.1.4 External Date and Time Formats

A collection of functions convert from the `psTime` types to various external formats. Note that ISO8601 format is "YYYY-MM-DDThh:mm:ss,Z"

```
psF64 psTimeToJD(const psTime *time);
psF64 psTimeToMJD(const psTime *time);
char *psTimeToISOTime(const psTime *time);
timeval *psTimeToTimeval(const psTime *time);
```

5.1.5 Date and Time Parsing

A related collection of functions convert from external formats to a `psTime` type.

```
psTime *psTimeFromJD(psF64 input);
psTime *psTimeFromMJD(psF64 input);
psTime *psTimeFromISO(const char *input);
psTime *psTimeFromTimeval(const timeval *input);
```

5.1.6 Date and Time Math

```
psTime *psTimeAdd(const psTime *tai1, psF64 delta);
psF64 psTimeSubtract(const psTime *time1, const psTime *time2);
psF64 psTimeDelta(const psTime *time1, const psTime *time2);
```

`psTimeAdd` adds some time to a `psTime`. `psTimeSubtract` subtracts two times. `psTimeDelta` gives the absolute time difference between two times.

Time math is only done on the `psTime` TAI type.

5.1.7 Time Tables

The offset of UTC from UT1, $\Delta\text{UT} = \text{UT1} - \text{UTC}$, as well as the polar motion, x_p and y_p , may be determined from table lookups. Tables are available covering different time periods and with different time resolution, and so it is important to be able to utilise multiple tables. Some tables may be found at:

- <ftp://maia.usno.navy.mil/ser7/ser7.dat>
- <ftp://maia.usno.navy.mil/ser7/finals.all> with explanatory guide at <ftp://maia.usno.navy.mil/ser7/readme.finals>. See also the web page <http://maia.usno.navy.mil/>.
- <http://hpiers.obspm.fr/eoppc/eop/eopc01/eopc01.1900-2004> (contains estimates prior to 1972).

The tables shall reside on local disk in known locations (i.e., there is no need that they are downloaded from the internet and parsed by PSLib). The format of these files shall be simple, for speed in reading: Each line shall contain the MJD, x_p (in arcseconds), y_p (in arcseconds) and ΔUT (in seconds) separated by whitespace;

blank lines and lines commencing with the comment character (the hash symbol, #) as the first character shall be ignored. For example:

```
# Bulletin A, 9 September 2004.
# ftp://maia.usno.navy.mil/ser7/ser7.dat
# MJD      XP(")  YP(")  UT1-UTC(s)
53258      0.1715 0.4774 -0.45092
53259      0.1734 0.4757 -0.45039
53260      0.1753 0.4741 -0.45012
53261      0.1770 0.4724 -0.45015
53262      0.1787 0.4708 -0.45045
```

The location of these files, their priority order, and the “from” and “to” dates of applicability will be specified through metadata (§5.1.7.1).

When a value is required, the tables shall be checked in priority order to see if the date is within the range of applicability for the table. If a table is found that is applicable, then the appropriate value shall be derived from linear interpolation between the nearest entries in the table. If no table is found that is applicable, and the required date is later than those covered in the tables, a warning shall be generated, and a pre-determined formula shall be applied (§5.1.7.1). For dates prior to those covered in the tables, the function shall generate a warning and use pre-determined values (§5.1.7.1).

We define a structure, `psTimeTable` to hold a single time table:

```
typedef struct {
    const char *filename; // Filename of time table
    const psF64 validFrom, validTo; // MJDs of validity of time table
    bool loaded; // Has the table been loaded?
    psVector *mjd; // MJD data, type is psF64
    psVector *dut; // delta UT = UT1 - UTC data
    psVector *xp, *yp; // Polar motion data
} psTimeTable;
```

The tables shall be read in only when required by the user (hence the `loaded` member); once loaded, the table shall remain in memory until the termination of the program. The corresponding constructor shall be:

```
psTimeTable *psTimeTableAlloc(const char *filename, psF64 validFrom, psF64 validTo);
```

which simply constructs a `psTimeTable` with the appropriate `filename`, `validFrom` and `validTo`. Upon construction, the `loaded` member shall be set to `false`, and the vectors containing the data shall be set to `NULL`.

The function `psTimeTableLoad` shall load a specified time table, returning `true` for success, and `false` if the table failed to load. If the table has already been loaded, then it shall be re-loaded.

```
bool psTimeTableLoad(psTimeTable *table);
```

Given a time table and a Modified Julian Date, `mjd`, at which to interpolate values, `psTimeTableInterpolate` shall calculate the appropriate UT1-UTC (`dut`; if non-`NULL`), and polar motion (`xp`, `yp`; if non-`NULL`) by interpolation on the table. If the `mjd` is outside the range of the table or the interpolation was for some other reason unsuccessful, then the function shall return `false`; otherwise the function shall set those parameters which are non-`NULL` (`dut`, `xp`, `yp`) and return `true`. If the requested `mjd` lies in the range of the table, but the table has not been loaded, then the function shall call `psTimeTableLoad`.

```
bool psTimeTableInterpolate(const psTimeTable *table, double *xp,
                           double *yp, double *dut, psF64 mjd);
```

5.1.7.1 Time Metadata

The following metadata keys will be used in time calculations:

Metadata key	Purpose
psLib.time.tables.dir	Time table directory
psLib.time.tables.files	Time table file names (space-delimited)
psLib.time.tables.from	Time tables are valid from these MJDs (vector)
psLib.time.tables.to	Time tables are valid to these MJDs (vector)
psLib.time.before.xp	Value of XP for before the earliest MJD
psLib.time.before.yp	Value of YP for before the earliest MJD
psLib.time.before.dut	Value of UT1-UTC for before the earliest MJD
pslib.time.predict.xp	A vector containing the x_p prediction formula coefficients
pslib.time.predict.yp	A vector containing the y_p prediction formula coefficients
pslib.time.predict.mjd	A value containing the MJD offset for temporary variables
pslib.time.predict.dut	A vector containing the UT1-UTC prediction formula coefficients

These metadata keys shall reside in a configuration file (§5.2.4), `psTime.config`, which shall be loaded into a `psMetadata` structure private to `psTime`, as part of `psLibInit`. An example of `psTime.config` follows:

```
# This configuration file specifies values required for time calculations by psLib.
psLib.time.tables.dir    STR    /home/panstarrs/psLib/config/          # Directory for time tables
psLib.time.tables.n      U8      2                                  # Number of time tables
psLib.time.tables.files  STR    bulletinA_09Sep2004.dat eopc01_1900_2004.40.dat # These are the file names
@psLib.time.tables.from  F64      53258.0, 15020.0                    # Valid from these MJDs
@psLib.time.tables.to    F64      53622.0, 53258.0                    # Valid to these MJDs
psLib.time.before.xp     F64      0.0                                # Value of XP for before the earliest MJD
psLib.time.before.yp     F64      0.0                                # Value of YP for before the earliest MJD
psLib.time.before.dut    F64      0.0                                # Value of UT1-UTC for before the earliest MJD

# Now follows formulae for predicting ahead of the most recent available table entry.
# xp = [0] + [1]*cos A + [2]*sin A + [3]*cos C + [4]*sin C
# yp = [0] + [1]*cos A + [2]*sin A + [3]*cos C + [4]*sin C
# A = 2*pi*(MJD - pslib.time.predict.mjd)/365.25
# C = 2*pi*(MJD - pslib.time.predict.mjd)/435.0
# dut = ut1-utc = [0] + [1]*(MJD - [2]) - (UT2-UT1)
# ut2-ut1 = 0.022 sin(2*pi*T) - 0.012 cos(2*pi*T) - 0.006 sin(4*pi*T) + 0.007 cos(4*pi*T)
# T = 2000.0 + (MJD - 51544.03)/365.2422
@psLib.time.predict.xp   F64      0.0569, 0.0555, -0.0200, 0.0513, 0.1483
@psLib.time.predict.yp   F64      0.3498, -0.0176, -0.0498, 0.1483, -0.0513
psLib.time.predict.mjd   F64      53257.0
@psLib.time.predict.dut  F64      -0.4944, -0.00023, 53262.0
```

5.2 Metadata

5.2.1 Conceptual Overview

Within PSLib, we provide a data structure to carry metadata and mechanisms to manipulate the metadata. Metadata is a general concept that requires some discussion. In any data analysis task, the ensemble of all possible data may be divided into two or three classes: there is the specific data of interest, there is data which is related or critical but not the primary data of interest, and there is all of the other data which may or may not be interesting. For example, consider a simple 2D image obtained of a galaxy from a CCD camera on a