DRAFT

Format for the Oceanographic and Atmospheric Master Library (OAML) Tide Constituent Database

Jan C. Depner
(depnerj@navo.navy.mil)
Code N43T
U.S. Naval Oceanographic Office
6 June 2003

DRAFT

Purpose

The purpose of this binary database is to replace the ASCII/XML formatted harmonic constituent data files, used in Dave Flater's (http://www.flaterco.com/xtide/) open-source XTide program, with a fast, efficient binary format. In addition, we want to replace the Naval Oceanographic Office's (http://www.navo.navy.mil) antiquated ASCII format harmonic constituent data file due to problems with configuration management of the file. The resulting database will become a Navy OAML (Oceanographic and Atmospheric Master Library) standard harmonic tide constituent database.

Design

The following describes the database file and some of the rationale behind the design. First question - Why didn't I use PostgreSQL or MySQL? Mostly portability. What? PostgreSQL runs on everything! Yes, but it doesn't come installed on everything. This would have meant that the poor, benighted Micro$oft borgs would have had to actually load a software package. In addition, the present harmonics/offset files only contain a total of 6,409 stations. It hardly seemed worth the effort (or overhead) of a full-blown RDBMS to handle this. Second question - Why binary and not ASCII or XML? This actually gets into philosophy. At NAVO we have used an ASCII harmonic constituent file for years (we were founded in 1830 and I think that that's when they designed the file). We have about fifty million copies floating around and each one is slightly different. Why? Because they're ASCII and everyone thinks they know what they are doing so they tend to modify the file. Same problem with XML, it's still ASCII. We wanted a file that people weren't going to mess with and that we could version control. We also wanted a file that was small and fast. This is very difficult to do with ASCII. The big slowdown with the old format was I/O and parsing. Third question - will it run on any OS? Hopefully, yes. After twenty-five years of working with low bidder systems I've worked on almost every OS known to man. Once you've been bitten by big endian vs little endian or IEEE floating point format vs VAX floating point format or byte addressable memory vs word addressable memory or 32 bit word vs 36 bit word vs 48 bit word vs 64 bit word sizes you get the message. All of the data in the file is stored either as ASCII text or scaled integers (32 bits or smaller), bit-packed and stuffed into an unsigned character buffer for I/O. No endian issues, no floating point issues, no word size issues, no memory mapping issues. I will be testing this on x86 Linux, HP-UX, and Micro$oft Windoze.

Now, on to the file layout. As much as I dislike ASCII it is occasionally handy to be able to see some information about a file without having to resort to a special purpose program. With that in mind I made the first part of the header of the file ASCII. The following is an example of the ASCII portion of the header:

[VERSION] = PFM Software - tide_db V1.07 - 2003-03-27
[LAST MODIFIED] = Mon Mar 31 20:09:11 2003
[HEADER SIZE] = 4096
[NUMBER OF RECORDS] = 11289
[START YEAR] = 1970
[NUMBER OF YEARS] = 68
[SPEED BITS] = 31
[SPEED SCALE] = 10000000
[SPEED OFFSET] = -410667
[EQUILIBRIUM BITS] = 16
[EQUILIBRIUM SCALE] = 100
[EQUILIBRIUM OFFSET] = 0
[NODE BITS] = 15
[NODE SCALE] = 10000
[NODE OFFSET] = -3949
[AMPLITUDE BITS] = 19
[AMPLITUDE SCALE] = 10000
[EPOCH BITS] = 16
[EPOCH SCALE] = 100
[RECORD TYPE BITS] = 4
[LATITUDE BITS] = 25
[LATITUDE SCALE] = 100000
[LONGITUDE BITS] = 26
[LONGITUDE SCALE] = 100000
[RECORD SIZE BITS] = 12
[STATION BITS] = 18
[DATUM OFFSET BITS] = 28
[DATUM OFFSET SCALE] = 10000
[DATE BITS] = 27
[MONTHS ON STATION BITS] = 10
[CONFIDENCE VALUE BITS] = 4
[TIME BITS] = 13
[LEVEL ADD BITS] = 17
[LEVEL ADD SCALE] = 1000
[LEVEL MULTIPLY BITS] = 16
[LEVEL MULTIPLY SCALE] = 1000
[DIRECTION BITS] = 9
[LEVEL UNIT BITS] = 3
[LEVEL UNIT TYPES] = 5
[LEVEL UNIT SIZE] = 15
[DIRECTION UNIT BITS] = 2
[DIRECTION UNIT TYPES] = 3
[DIRECTION UNIT SIZE] = 15
[RESTRICTION BITS] = 4
[RESTRICTION TYPES] = 2
[RESTRICTION SIZE] = 30
[PEDIGREE BITS] = 7
[PEDIGREE TYPES] = 14

[PEDIGREE SIZE] = 70
[DATUM BITS] = 7
[DATUM TYPES] = 61
[DATUM SIZE] = 70
[CONSTITUENT BITS] = 8
[CONSTITUENTS] = 173
[CONSTITUENT SIZE] = 10
[TZFILE BITS] = 10
[TZFILES] = 408
[TZFILE SIZE] = 30
[COUNTRY BITS] = 9
[COUNTRIES] = 240
[COUNTRY SIZE] = 50
[END OF FILE] = 2759729
[END OF ASCII HEADER DATA]

Most of these values will make sense in the context of the following description of the rest of the file. Some caveats on the data storage - if no SCALE is listed for a field, the scale is 1. If no BITS field is listed, this is a variable length character field and is stored as 8 bit ASCII characters. If no OFFSET is listed, the offset is 0. Offsets are scaled. All SIZE fields refer to the maximum length, in characters, of a variable length character field. Some of the BITS fields are calculated while others are hardwired (see code). For instance, [DIRECTION BITS] is hardwired because it is an integer field whose value can only be from 0 to 361 (361 = no direction flag). [NODE BITS], on the other hand, is calculated on creation by checking the min, max, and range of all of the node factor values. The number of bits needed is easily calculated by taking the log of the adjusted, scaled range, dividing by the log of 2 and adding 1. Immediately following the ASCII portion of the header is a 32 bit checksum of the ASCII portion of the header. Why? Because someone will get the idea that he/she can modify the header with a text or hex editor. Go figure.

The rest of the header is as follows :

[LEVEL UNIT TYPES] fields of [LEVEL UNIT SIZE] characters, each field is internally 0 terminated

[DIRECTION UNIT TYPES] fields of [DIRECTION UNIT SIZE] characters, 0 terminated

[RESTRICTION TYPES] fields of [RESTRICTION SIZE] characters, 0 terminated

[PEDIGREE TYPES] fields of [PEDIGREE SIZE] characters, 0 terminated

[TZFILES] fields of [TZFILE SIZE] characters, 0 terminated

[COUNTRIES] fields of [COUNTRY SIZE] characters, 0 terminated

[DATUM TYPES] fields of [DATUM SIZE] characters, 0 terminated

[CONSTITUENTS] fields of [CONSTITUENT SIZE] characters, 0 terminated. Yes, I know, I wasted some space with these fields but I wasn't worried about a couple of hundred bytes.

[CONSTITUENTS] fields of [SPEED BITS], speed values (scaled and offset)

[CONSTITUENTS] groups of [NUMBER OF YEARS] fields of [EQUILIBRIUM BITS], equilibrium arguments (scaled and offset)

[CONSTITUENTS] groups of [NUMBER OF YEARS] fields of [NODE BITS], node factors (scaled and offset)

Finally, the data. At present there are two types of records in the file. These are reference stations (record type 1) and subordinate stations (record type 2). Reference stations contain a set of constituents while subordinate stations contain a number of offsets to be applied to the reference station that they are associated with. Note that reference stations (record type 1) may, in actuality, be subordinate stations, albeit with a set of constituents. All of the records have the following subset of information stored as the first part of the record:

[RECORD SIZE BITS] - record size in bytes
[RECORD TYPE BITS] - record type (1 or 2)
[LATITUDE BITS] - latitude (degrees, south negative, scaled & offset)
[LONGITUDE BITS] - longitude (degrees, west negative, scaled & offset)
[TZFILE BITS] - index into timezone array (retrieved from header)
variable size - station name, 0 terminated
[STATION BITS] - record number of reference station or -1
[COUNTRY_BITS] index into country array (retrieved from header)
[PEDIGREE BITS] - index into pedigree array (retrieved from header)
variable size - source, 0 terminated
[RESTRICTION BITS] - index into restriction array
variable size - comments, may contain LFs to indicate newline (no CRs)

These are the rest of the fields for record type 1:

[LEVEL UNIT BITS] - index into level units array

[DATUM OFFSET BITS] - datum offset (scaled)
[DATUM BITS] - index into datum name array
[TIME BITS] - time zone offset from GMT0 (meridian, integer
+/-HHMM)
[DATE BITS] - expiration date, (integer YYYYMMDD, default is 0)
[MONTHS ON STATION BITS] - months on station
[DATE BITS] - last date on station, default is 0
[CONFIDENCE BITS] - confidence value (TBD)
[CONSTITUENT BITS] - "N", number of constituents for this station

N groups of:
[CONSTITUENT BITS] - constituent number
[AMPLITUDE BITS] - amplitude (scaled & offset)
[EPOCH BITS] - epoch (scaled & offset)

These are the rest of the fields for record type 2:

[LEVEL UNIT BITS] - leveladd units, index into level_units array
[DIRECTION UNIT BITS] - direction units, index into
dir_units array
[LEVEL UNIT BITS] - avglevel units, index into level_units array
[TIME BITS] - min timeadd (integer +/-HHMM) or NULL (0)
[LEVEL ADD BITS] - min leveladd (scaled) or NULL (0)
[LEVEL MULTIPLY BITS] - min levelmultiply (scaled) or NULL (0)
[LEVEL ADD BITS] - min avglevel (scaled) or NULL (0)
[DIRECTION BITS] - min direction (0-360 or NULL (361))
[TIME BITS] - max timeadd (integer +/-HHMM) or NULL (0)
[LEVEL ADD BITS] - max leveladd (scaled) or NULL (0)
[LEVEL MULTIPLY BITS] - max levelmultiply (scaled) or NULL (0)
[LEVEL ADD BITS] - max avglevel (scaled) or NULL (0)
[DIRECTION BITS] - max direction (0-360 or NULL (361))
[TIME BITS] - floodbegins (integer +/-HHMM) or NULL (0xA00)
[TIME BITS] - ebbbegins (integer +/-HHMM) or NULL (0xA00)

The NULL values referenced above should be handled in the following manner by applications:

- For timeadd, the "right thing" is to treat the difference as zero anyway.
- For levelmultiply, the "right thing" is to treat the value as unity (multiply by one, no change).
- Directions specify 361 as the "null" value. For those fields, zero means zero.
- Slack offsets (floodbegins, ebbbegins) are somewhat complicated. It is not correct to default slack offsets to zero when they are omitted. Consider an example where max timeadd is 6 hours, min timeadd is 5 hours, and slack offsets are omitted. If the slack offsets are defaulted to zero, the results are nonsense. Instead, the slack offsets must be defaulted to values that are

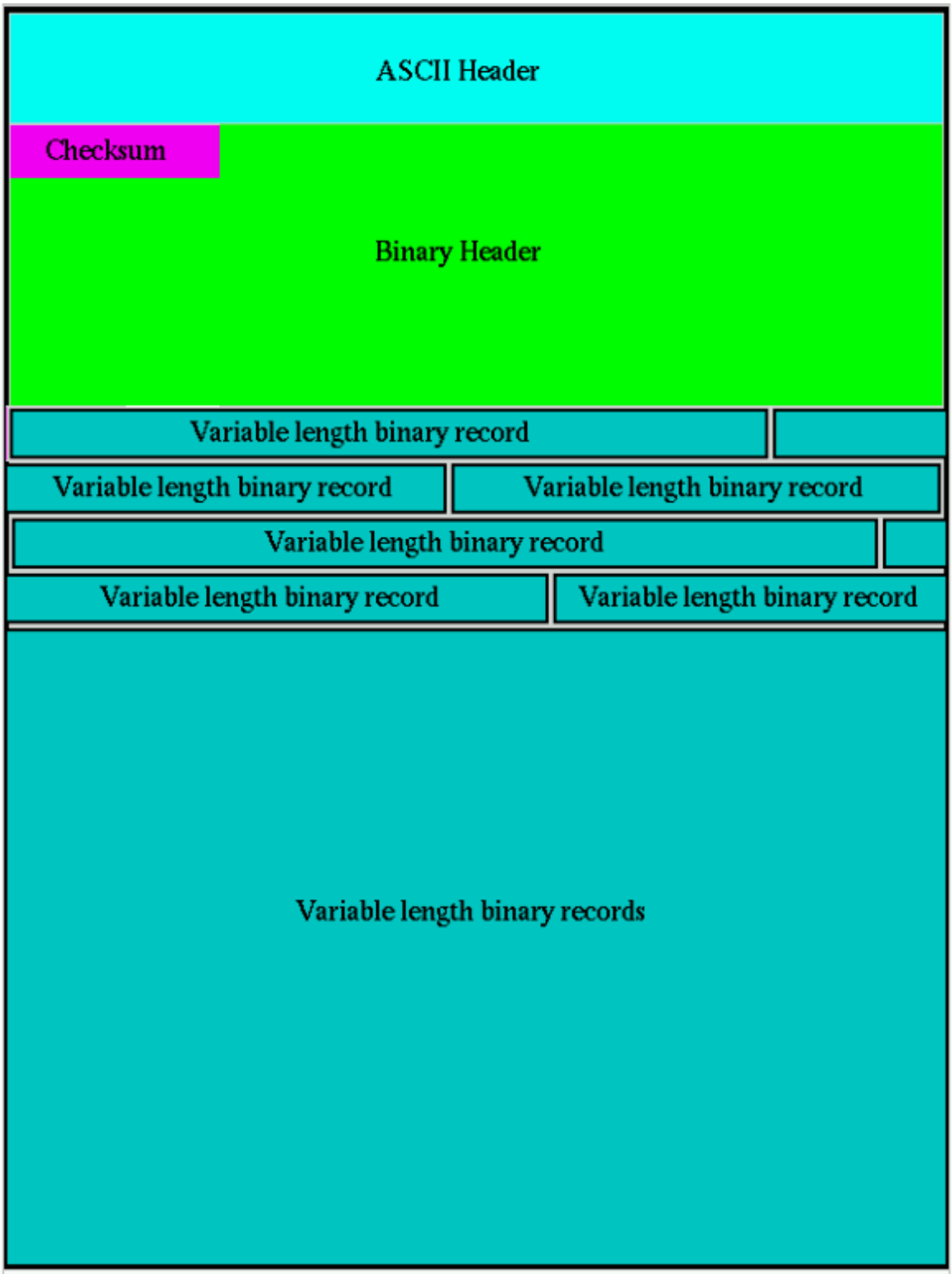
"reasonable" given the skew of the max and min offsets. XTide simply averages the two. Omitted slack offsets are represented with the value NULLSLACKOFFSET defined in libtcd/tcd.h, which is 0xA00 (2560).

The following is a clarification from Dave Flater on the original “simplified” type 2 records:

Prior to the revisions of 2003-03-27, a simplified representation could be used for type 2 records where the max and min values were identical and there were no slack offsets, current directions, or average levels. In such cases, the max fields were all set to their null values and only the min fields were used. This corresponds to a shorthand notation that is used in the offsets.xml format. It is also analogous to a certain implementation detail in XTide, where a simpler algorithm can be used for simple offsets. However, XTide does its own simplifications and its behavior is not impacted by the choice of representation in the database.

Unfortunately, "simplified" type 2 records also caused a conflict. A station having a nonzero min time offset and a zero max time offset with no other fields being relevant would be mistaken for a simplified case, and the min time offset would incorrectly be used for both max and min. This conflict was resolved in the 2003-03-27 revision of libtcd, tcd-utils, and XTide by doing away with "simplified" type 2 records (which were undocumented to begin with).

The following figure is a simple diagram of the file layout:



Back to philosophy! When you design a database of any kind the first thing you should ask yourself is "Self, how am I going to access this data most of the time?". If you answer yourself out loud you should consider seeing a shrink. 99 and 44/100ths percent of the time this database is going to be read to get station data. The other 66/100ths percent of the time it will be created/modified. Variable length records are no problem on retrieval. They are no problem to create. They can be a major pain in the backside if you have to modify/delete them. Since we shouldn't be doing too much editing of the data (usually just adding records) this is a pretty fair design. At some point though we are going to want to modify or delete a record. There are two possibilities here. We can dump the database to an ASCII file or files using restore_tide_db, use a text editor to modify them, and then rebuild the database. The other possibility is to modify the record in place. This is OK if you don't change a variable length field but what if you want to change the station name or add a couple of constituents? With the design as is we have to read the remainder of the file from the end of the record to be modified, write the modified record, rewrite the remainder of the file, and then change the end_of_file pointer in the header. So, which fields are going to be a problem? Changes to station name, source, comments, or the number of constituents for a station will require a resizing of the database. Changes to any of the other fields can be done in place. The worst thing that you can do though is to delete a record. Not just because the file has to be resized but because it might be a reference record with subordinate stations. These would have to be deleted as well. The delete_tide_record function will do just that so make sure you check before you call it. You might not want to do that.

Another point to note is that when you open the database the records are indexed at that point. This takes about half a second on a dual 450. Most applications use the header part of the record very often and the rest of the record only if they are going to actually produce predicted tides. For instance, XTide plots all of the stations on a world map or globe and lists all of the station names. It also needs the timezone up front. To save re-indexing to get these values I save them in memory. The only time an application needs to actually read an entire record is when you want to do the prediction. Otherwise just use get_partial_tide_record or get_next_partial_tide_record to yank the good stuff out of memory.

Application Programming Interface

The following describes the public data structures and each public function call. Note that all of the functions in the API use the NAVOCEANO standard data types as defined in nvtypes.h in order to limit problems based on architecture.

Data structures

```
typedef struct
{
    NV_CHAR           version[80];
    NV_CHAR           last_modified[30];
    NV_INT32          number_of_records;
    NV_INT32          start_year;
    NV_INT32          number_of_years;
    NV_INT32          constituents;
    NV_INT32          level_unit_types;
    NV_INT32          dir_unit_types;
    NV_INT32          restriction_types;
    NV_INT32          pedigree_types;
    NV_INT32          datum_types;
    NV_INT32          countries;
    NV_INT32          tzfiles;
} DB_HEADER_PUBLIC;


/*  Header portion of each station record.  */

typedef struct
{
    NV_INT32               record_number;
    NV_INT32               record_size;
    NV_U_BYTE              record_type;
    NV_FLOAT64             latitude;
    NV_FLOAT64             longitude;
    NV_INT32               reference_station;
    NV_INT16               tzfile;
    NV_CHAR                name[NAME_LENGTH];
} TIDE_STATION_HEADER;
```

```
/*  Tide station record.  */

typedef struct
{
    TIDE_STATION_HEADER     header;
    NV_INT16                country;
    NV_INT16                pedigree;
    NV_CHAR                 source[SOURCE_LENGTH];
    NV_U_BYTE               restriction;
    NV_CHAR                 comments[COMMENTS_LENGTH];
    NV_U_BYTE               units;
    NV_FLOAT32              datum_offset;
    NV_INT16                datum;
    NV_INT32                zone_offset;
    NV_U_INT32              expiration_date;
    NV_U_BYTE               months_on_station;
    NV_U_INT32              last_date_on_station;
    NV_U_BYTE               confidence;
    NV_FLOAT32              amplitude[MAX_CONSTITUENTS];
    NV_FLOAT32              epoch[MAX_CONSTITUENTS];
    NV_U_BYTE               level_units;
    NV_U_BYTE               avg_level_units;
    NV_U_BYTE               direction_units;
    NV_INT32                min_time_add;
    NV_FLOAT32              min_level_add;
    NV_FLOAT32              min_level_multiply;
    NV_FLOAT32              min_avg_level;
    NV_INT32                min_direction;
    NV_INT32                max_time_add;
    NV_FLOAT32              max_level_add;
    NV_FLOAT32              max_level_multiply;
    NV_FLOAT32              max_avg_level;
    NV_INT32                max_direction;
    NV_INT32                flood_begins;
    NV_INT32                ebb_begins;
} TIDE_RECORD;
```

Function Definitions

```
/*****************************************************************************\

    Function        dump_tide_record - prints out all of the fields in the
                    input tide record

    Synopsis        dump_tide_record (rec);

                    TIDE_RECORD *rec        pointer to the tide record

    Returns         void

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        get_country - gets the country field for record "num"

    Synopsis        get_country (num);

                    NV_INT32 num            tide record number

    Returns         NV_CHAR *               country name (associated with
                                            ISO 3166-1:1999 2-character
                                            country code

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        get_tzfile - gets the time zone name for record "num"

    Synopsis        get_tzfile (num);

                    NV_INT32 num            tide record number

    Returns         NV_CHAR *               time zone name used in TZ variable

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/
```

```
/*****************************************************************************\

    Function        get_station - get the name of the station for record "num"

    Synopsis        get_station (num);

                    NV_INT32 num            tide record number

    Returns         NV_CHAR *               station name

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        get_constituent - get the constituent name for constituent
                    number "num"

    Synopsis        get_constituent (num);

                    NV_INT32 num            constituent number

    Returns         NV_CHAR *               constituent name

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        get_level_units - get the level units for level units
                    number "num"

    Synopsis        get_level_units (num);

                    NV_INT32 num            level units number

    Returns         NV_CHAR *               units (ex. "meters");

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/
```

```
/*****************************************************************************\

    Function        get_dir_units - get the direction units for direction
                    units number "num"

    Synopsis        get_dir_units (num);

                    NV_INT32 num            direction units number

    Returns         NV_CHAR *               units (ex. "degrees true");

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        get_restriction - gets the restriction description for
                    restriction number "num"

    Synopsis        get_restriction (num);

                    NV_INT32 num            restriction number

    Returns         NV_CHAR *               restriction (ex. "PUBLIC DOMAIN");

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        get_pedigree - gets the pedigree description for pedigree
                    number "num"

    Synopsis        get_pedigree (num);

                    NV_INT32 num            pedigree number

    Returns         NV_CHAR *               pedigree description

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/
```

```
/*****************************************************************************\

    Function        get_datum - gets the datum name for datum number "num"

    Synopsis        get_datum (num);

                    NV_INT32 num            datum number

    Returns         NV_CHAR *               datum name

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        get_speed - gets the speed value for constituent number
                    "num"

    Synopsis        get_speed (num);

                    NV_INT32 num            constituent number

    Returns         NV_FLOAT64              speed

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        get_equilibrium - gets the equilibrium value for
                    constituent number "num" and year "year"

    Synopsis        get_equilibrium (num, year);

                    NV_INT32 num            constituent number
                    NV_INT32 year           year

    Returns         NV_FLOAT32              equilibrium argument

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/
```

```
/********************************************************************\

    Function        get_node_factor - gets the node factor value for
                    constituent number "num" and year "year"

    Synopsis        get_node_factor (num, year);

                    NV_INT32 num            constituent number
                    NV_INT32 year           year

    Returns         NV_FLOAT32              node factor

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/


/********************************************************************\

    Function        get_partial_tide_record - gets "header" portion of record
                    "num" from the index that is stored in memory.  This is
                    way faster than reading it again and we have to read it
                    to set up the index.  This costs a bit in terms of
                    memory but most applications use this data far more than
                    the rest of the record.

    Synopsis        get_partial_tide_record (num, rec);

                    NV_INT32 num              record number
                    TIDE_STATION_HEADER *rec  header portion of the record

    Returns         NV_BOOL                   NVTrue if successful

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/


/********************************************************************\

    Function        get_next_partial_tide_record - gets "header" portion of
                    the next record from the index that is stored in memory.

    Synopsis        get_next_partial_tide_record (rec);

                    TIDE_STATION_HEADER *rec  header portion of the record

    Returns         NV_INT32                  record number or -1 on failure

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/
```

```
/********************************************************************\

    Function        get_nearest_partial_tide_record - gets "header" portion of
                    the record closest geographically to the input position.

    Synopsis        get_nearest_partial_tide_record (lat, lon, rec);

                    NV_FLOAT64 lat            latitude
                    NV_FLOAT64 lon            longitude
                    TIDE_STATION_HEADER *rec  header portion of the record

    Returns         NV_INT32                  record number or -1 on failure

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/


/********************************************************************\

    Function        get_time - converts a time string in +/-HH:MM form to an
                    integer in +/-HHMM form

    Synopsis        get_time (string);

                    NV_CHAR *string         time string

    Returns         NV_INT32                time

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/


/********************************************************************\

    Function        ret_time - converts a time value in +/-HHMM form to a
                    time string in +/-HH:MM form

    Synopsis        ret_time (time);

                    NV_INT32                time

    Returns         NV_CHAR *               time string

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/
```

```
/********************************************************************\

    Function        get_tide_db_header - gets the public portion of the tide
                    database header

    Synopsis        get_tide_db_header ();

    Returns         DB_HEADER_PUBLIC        public tide header

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/


/********************************************************************\

    Function        search_station - returns record numbers of all stations
                    that have the string "string" anywhere in the station
                    name.  This search is case insensitive.  When no more
                    records are found it returns -1;

    Synopsis        search_station (string);

                    NV_CHAR *string         search string

    Returns         NV_INT32                record number or -1 when no more
                                            matches

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/


/********************************************************************\

    Function        find_station - finds the record number of the station
                    that has name "name"

    Synopsis        find_station (name);

                    NV_CHAR *name           station name

    Returns         NV_INT32                record number

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/
```

```
/************************************************************************\

    Function        find_tzfile - gets the timezone number (index into
                    tzfile array) given the tzfile name

    Synopsis        find_tzfile (tname);

                    NV_CHAR *tname          tzfile name

    Returns         NV_INT32                tzfile number

    Author          Jan C. Depner
    Date            08/01/02

\************************************************************************/


/************************************************************************\

    Function        find_country - gets the timezone number (index into
                    country array) given the country name

    Synopsis        find_country (tname);

                    NV_CHAR *tname          country name

    Returns         NV_INT32                country number

    Author          Jan C. Depner
    Date            08/01/02

\************************************************************************/


/************************************************************************\

    Function        find_level_units - gets the index into the level_units
                    array given the level units name

    Synopsis        find_level_units (tname);

                    NV_CHAR *tname          units name (ex. "meters")

    Returns         NV_INT32                units number

    Author          Jan C. Depner
    Date            08/01/02

\************************************************************************/
```

```
/*****************************************************************************\

    Function        find_dir_units - gets the index into the dir_units
                    array given the direction units name

    Synopsis        find_dir_units (tname);

                    NV_CHAR *tname          units name (ex. "degrees true")

    Returns         NV_INT32                units number

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        find_pedigree - gets the index into the pedigree array
                    given the pedigree name

    Synopsis        find_pedigree (tname);

                    NV_CHAR *tname          pedigree name

    Returns         NV_INT32                pedigree number

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        find_datum - gets the index into the datum array given the
                    datum name

    Synopsis        find_datum (tname);

                    NV_CHAR *tname          datum name

    Returns         NV_INT32                datum number

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/
```

```
/*****************************************************************************\

    Function        find_constituent - gets the index into the constituent
                    arrays for the named constituent.

    Synopsis        find_constituent (name);

                    NV_CHAR *name           constituent name (ex. M2)

    Returns         NV_INT32                index into constituent arrays or -1
                                            on failure

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        find_restriction - gets the index into the restriction
                    array given the restriction name

    Synopsis        find_restriction (tname);

                    NV_CHAR *tname          restriction name

    Returns         NV_INT32                restriction number

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        set_speed - sets the speed value for constituent "num"

    Synopsis        set_speed (num, value);

                    NV_INT32 num            constituent number
                    NV_FLOAT64 value        speed value

    Returns         void

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/
```

```
/*****************************************************************************\

    Function        set_equilibrium - sets the equilibrium argument for
                    constituent "num" and year "year"

    Synopsis        set_equilibrium (num, year, value);

                    NV_INT32 num            constituent number
                    NV_INT32 year           year
                    NV_FLOAT64 value        equilibrium argument

    Returns         void

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        set_node_factor - sets the node factor for constituent
                    "num" and year "year"

    Synopsis        set_node_factor (num, year, value);

                    NV_INT32 num            constituent number
                    NV_INT32 year           year
                    NV_FLOAT64 value        node factor

    Returns         void

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        add_pedigree - adds a new pedigree to the database

    Synopsis        add_pedigree (tname, db);

                    NV_CHAR *tname          new pedigree string
                    DB_HEADER_PUBLIC *db    modified header

    Returns         NV_INT32                new pedigree index

    Author          Jan C. Depner
    Date            09/20/02

\*****************************************************************************/
```

```
/*****************************************************************************\

    Function        add_tzfile - adds a new tzfile to the database

    Synopsis        add_tzfile (tname, db);

                    NV_CHAR *tname          new tzfile string
                    DB_HEADER_PUBLIC *db    modified header

    Returns         NV_INT32                new tzfile index

    Author          Jan C. Depner
    Date            09/20/02

\*****************************************************************************/


/*****************************************************************************\

    Function        add_country - adds a new country to the database

    Synopsis        add_country (tname, db);

                    NV_CHAR *tname          new country string
                    DB_HEADER_PUBLIC *db    modified header

    Returns         NV_INT32                new country index

    Author          Jan C. Depner
    Date            09/20/02

\*****************************************************************************/


/*****************************************************************************\

    Function        add_datum - adds a new datum to the database

    Synopsis        add_datum (tname, db);

                    NV_CHAR *tname          new datum string
                    DB_HEADER_PUBLIC *db    modified header

    Returns         NV_INT32                new datum index

    Author          Jan C. Depner
    Date            09/20/02

\*****************************************************************************/
```

```
/********************************************************************\

    Function        add_restriction - adds a new restriction to the database

    Synopsis        add_restriction (tname, db);

                    NV_CHAR *tname          new restriction string
                    DB_HEADER_PUBLIC *db    modified header

    Returns         NV_INT32                new restriction index

    Author          Jan C. Depner
    Date            09/20/02

\********************************************************************/


/********************************************************************\

    Function        check_simple - checks tide record to see if it is a
                    "simple" subordinate station.

    Synopsis        check_simple (rec);

                    TIDE_RECORD rec         tide record

    Returns         NV_BOOL                 NVTrue if "simple"

    Author          Jan C. Depner
    Date            08/01/02

    Modified        David Flater
    Date            2003-03-27

    "Simplified" type 2 records were done away with 2003-03-27 per the
    discussion in http://www.flaterco.com/xtide/tcd_notes.html.  This
    function is now only a convenience to check whether the record
    *could* be simplified, and is used only by restore_tide_db to
    determine whether it should output a shorthand XML notation.

\********************************************************************/


/********************************************************************\

    Function        open_tide_db - opens the tide database

    Synopsis        open_tide_db (file);

                    NV_CHAR *file           database file name

    Returns         NV_BOOL                 NVTrue if file opened

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/
```

```
/********************************************************************\

    Function        close_tide_db - closes the tide database

    Synopsis        close_tide_db ();

    Returns         void

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/


/********************************************************************\

    Function        create_tide_db - creates the tide database

    Synopsis        create_tide_db (file, constituents, constituent, speed,
                        start_year, num_years, equilibrium, node_factor);

                    NV_CHAR *file               database file name
                    NV_INT32 constituents       number of constituents
                    NV_CHAR *constituent[]      constituent names
                    NV_FLOAT64 *speed           speed values
                    NV_INT32 start_year         start year
                    NV_INT32 num_years          number of years
                    NV_FLOAT32 *equilibrium[]   equilibrium arguments
                    NV_FLOAT32 *node_factor[]   node factors

    Returns         NV_BOOL                 NVTrue if file created

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/


/********************************************************************\

    Function        write_tide_record - writes a tide record to the database

    Synopsis        write_tide_record (num, rec);

                    NV_INT32 num            record number, -1 for a new record
                    TIDE_RECORD *rec        tide record

    Returns         NV_BOOL                 NVTrue if successful

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/
```

```
/********************************************************************\

    Function        read_next_tide_record - reads the next tide record from
                    the database

    Synopsis        read_next_tide_record (rec);

                    TIDE_RECORD *rec        tide record

    Returns         NV_INT32                record number of the tide record

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/


/********************************************************************\

    Function        read_tide_record - reads tide record "num" from the
                    database

    Synopsis        read_tide_record (num, rec);

                    NV_INT32 num            record number
                    TIDE_RECORD *rec        tide record

    Returns         NV_INT32                record number of the tide record

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/


/********************************************************************\

    Function        add_tide_record - adds a tide record to the database

    Synopsis        add_tide_record (rec);

                    TIDE_RECORD *rec        tide record

    Returns         NV_BOOL                 NVTrue if successful

    Author          Jan C. Depner
    Date            08/01/02

\********************************************************************/
```

```
/*****************************************************************************\

    Function        delete_tide_record - deletes a record and all subordinate
                    records from the database

    Synopsis        delete_tide_record (num);

                    NV_INT32 num            record number

    Returns         void                    NVTrue if successful

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/


/*****************************************************************************\

    Function        update_tide_record - updates a tide record in the database

    Synopsis        update_tide_record (num, rec);

                    NV_INT32 num            record number
                    TIDE_RECORD *rec        tide record

    Returns         NV_BOOL                 NVTrue if successful

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/
```

```
/*****************************************************************************\

    Function        infer_constituents - computes inferred constituents when
                    M2, S2, K1, and O1 are given.  This function fills the
                    remaining unfilled constituents.  The inferred constituents
                    are developed or decided based on article 230 of
                    "Manual of Harmonic Analysis and Prediction of Tides",
                    Paul Schureman, C & GS special publication no. 98,
                    October 1971.  This function is really just for NAVO
                    since we go to weird places and put in tide gages for
                    ridiculously short periods of time so we only get a
                    few major constituents developed.  This function was
                    modified from the NAVO FORTRAN program pred_tide_corr,
                    subroutine infer.ftn, 08-oct-86.

    Synopsis        infer_constituents (rec);

                    TIDE_RECORD rec         tide record

    Returns         NV_BOOL                 NVFalse if not enough constituents
                                            available to infer others

    Author          Jan C. Depner
    Date            08/01/02

\*****************************************************************************/
```