Specifying Multiple Calendars, Calendric Systems, and Field Tables and Functions in TimeADT

Nick Kline, Jie Li and Richard Snodgrass

May 28, 1999

TR-41

A TIMECENTER Technical Report

Title	Specifying Multiple Calendars, Calendric Systems, and Field Tables and Functions in TimeADT
Author(s)	Nick Kline, Jie Li and Richard Snodgrass
Publication History	May 1999. A TIMECENTER Technical Report.

TIMECENTER Participants

Abstract

TIMEADT provides multiple calendar support for C and C++ applications. We describe here the TIMEADT automatic generation tool which provides the capability of configuring the TIMEADT system with different calendars, calendric systems, properties. This tool takes calendar specification files and TIMEADT specification file and generates a C source file and a C header file which contain code for integrating calendars and calendric systems into TIMEADT (These generated files can also be used automatically by C++ classes in TimeADT).

Each calendar has a calendar specification file which defines the temporal granularities within the calendar and additional field names. The TIMEADT specification file defines the calendric systems used by the application, as well as the location of calendar specification files, initial property values, aliases for field value functions and tables, renaming and renumbering of granularities, mappings between granularities from different calendars, and declarations of a operating system time function and a mapping between operating system time unit and a particular granularity.

1 Introduction

This paper describes the TIMEADT automatic generation tool and file and data formats used by the tool. TIMEADT extends traditional relational database systems with support for using multiple calendars. Typically, only the Gregorian calendar is available within a relational database. Date calculations are complex with conversion of temporal constants (time-stamps or dates) to other calendar systems (such as the Julian calendar, the Hebrew calendar, or the Chinese Lunar calendar) complicated and awkward, if possible at all. Besides allowing multiple calendars to be utilized, TIMEADT supports the combination of calendars into *calendric systems*, which allow centralized, controlled processing of dates as well as reuse of calendars.

A calendric system might use a geologic calendar for very old time-stamps (such as dates older than 70 thousand years ago), a carbon-14 calendar for more recent dates (older than three thousand years ago but less than 70 thousand years old), a tree-ring calendar for more recent historical times (dates from 1000 BC to 1000 AD), the Julian calendar for times from 1000 AD to September 14, 1752, and the Gregorian calendar for time-stamps from September 14, 1752 into the future.

This tool integrates calendars into the context of the TIMEADT system. We assume that the reader is familiar with the MultiCal System [Soo & Snodgrass 1992]. Calendars are provided in the form of C source files with a Makefile or binaries for the target machine. Each calendar has a *calendar specification file*, used to describe the temporal granularities within the calendar and other information used to communicate between the Uniform Calendric System (UCS) and calendars. In addition, there is a special TIMEADT *specification file* which describes the organization of the system and initial values of system tables, such as the properties table.

The generation tools take the calendar specifications and calendric system specification and generates a C source file which contain the C code and C data structures to integrate calendars and calendric systems into TIMEADT and a C header file which exports a list of granularities. The generated C source and header files are given in Appendix A.2 and Appendix A.4 respectively, as well as a discussion of their specific contents.

This paper is intended for DBMS administrators who need to understand how to configure their calendric system specification file to incorporate the field value tables, functions and calendars they utilize. Calendar writer's may also be able to make use of this document as an aid to understanding how their calendars will be utilized in the scope of the TIMEADT system. Finally, the appendix will be of use in maintaining the TIMEADT generator.

2 The TIMEADT Specification File

In this section we describe the overall configuration of the TIMEADT System. The TIMEADT specification file contains the configuration information. It contains the location of the various calendar specification files, the description of the calendric systems (including their component calendars and epochs), the default calendric system, the default properties, the field value tables and functions, and the mappings between the granularities of different calendars. It is maintained by the database administrator.

2.1 An Example TIMEADT Specification File

Figure 1 shows an example TIMEADT specification file **example.spec**. This example imports two calendars (`Gregorian` and `Astronomy`), declares a calendric system (`american`), renames the granularity names, gives the initial property values, defines the field value functions and tables used, declares the mappings between the granularities of different calendars, renumbers the granularity external value, specifies default plausibility and semantics, declares two distributions, and declares operating system time function and mapping between granularity second and operating system time unit.

The file begins with a comment line. Any line beginning with a hash mark (#) is a comment line. Blank lines are ignored. Words are case sensitive.

The import statement supplies a local calendar name (use that name subsequently in this file only; elsewhere use the name given in the calendar specification file), as well as the path to the calendar. The path may be either relative to the location of the **example.spec** file or absolute. The calendar specification files must be in the directory given by the path and in the form of *cal_name*.spec. In the example **example.spec**, we require calendar files called **gregorian.spec** and **astronomy.spec**. The temporary calendar name is required to be a legal C identifier and the file path must be a legal file path under the implementation machine's operating system, containing no spaces or control characters.

Right after the import statements are rename statements. The names of granularities from different calendars may be identical. The purpose of this statement is to give each granularity a unique global identifier and the global identifiers will be used in granularity declaration and renumber statement in the **example.spec** file.

Following after the rename statements are the `Calendar epoch` statement which specifies the calendar used to parse the event constants. The calendar can be one of the imported calendars. The `Input format` statement is used to change the format in which these event constants are given. The default input format is below.

`<month_of_year,english_month_names>␣<day_of_month,arabic_numeral>,␣`
`<year,arabic_numeral>`

Events are used to specify when a particular calendar is used to process dates within a calendric system. The format with which these events are specified is given in the `input format` statement. The input format should be a valid event property string, parseable by the epoch calendar specified in the `calendar epoch` statement [Soo & Snodgrass 1992]. Parsing event constants is done in the initialization routine of TIMEADT program. Should parsing fails, the program will print out error message and halt. The next directive is the declaration of the the `american` calendric system. This calendric system is composed of two epochs. The calendric system uses an Astronomy calendar from the beginning of time to the chronon before September 14, 1752. Starting with this date the calendric system uses the Gregorian calendar. Implicitly, the last calendar in a calendar list has an epoch that extends to `'forever'`. The first calendar in a calendar list must begin with the `'beginning'`. To allow gaps in a calendric system, we introduce the nonexistent calendar, called `none` to fill the gaps. Immediately following the list of calendars is the input order for the calendars. This list specifies the order in which the UCS will try to parse temporal constants. A calendric

```
#TimeADT specification file from the University of Arizona TimeCenter project

Import Gregorian from ./gregorian
Import Astronomy from ./astronomy

Rename Astronomy.second as astro_second
Rename Astronomy.day_hundredth as astro_day_hundredth
Rename Astronomy.day as astro_day
Rename Astronomy.year as astro_year
Rename Astronomy.century as astro_century

Calendar epoch is Gregorian
Input format is '<month_of_year,english_month_names> <day_of_month,arabic_numeral>,
<year,arabic_numeral>'
Define calendric system american as  Astronomy 'beginning',
                                     Gregorian 'September 14, 1752'
  with default input order Gregorian, Astronomy

Property locale is 'Tucson'
Property instant_input_format is '<month_of_year,english_month_names>
<day_of_month,arabic_numeral>, <year,arabic_numeral>'
Property instant_output_format is '<month_of_year,english_month_names>
<day_of_month,arabic_numeral>, <year,arabic_numeral>'
Property interval_input_format is  '<month,arabic_numeral> months'
Property interval_output_format is '<month,arabic_numeral> months'
Property now_separator is ' '
Property period_input_separator is ' - '
Property period_output_separator is ' - '
Property period_output_delimiters is '[)'
Property indeterminacy_input_separator is ' ~ '
Property indeterminacy_output_separator is ' ~ '
Property default_input_distribution is 'uniform'
Property missing_distribution is 'missing'
Property distribution_input_format is '<value> with <distribution> distribution'
Property distribution_output_format is '<value> with <distribution> distribution'
Property override_input_epoch is 'Gregorian'
Property beginning is 'beginning'
Property forever is 'forever'
Property now is 'now'

Field value function arabic_numeral is
        ascii_arabic_cardinal_symbols from 0 to 4294967295
Field value table english_month_names
        is ascii_english_gregorian_month_names from 1 to 12
Field value table mandarin_month_names
is ascii_romanized_mandarin_gregorian_month_names from 1 to 12
Field value table danish_month_names
        is latex_danish_gregorian_month_names from 1 to 12

Granularity astro_second is 1 second
Renumber granularity day  as 11

Default plausability is 50
Default maximum number of iterations is 15
Default semantics is finest and cast
Distribution uniform is based on PMF_uniform function
Distribution pyramid is based on PMF_pyramid function

Operating system time function is sys_time
Granularity second is irregular operating system time unit with function unit_to_sec
```


Figure 1: Example **example.spec** file

system must contain at least one calendar. The `with default input order` list must also contain at least one calendar.

The default calendric system used in the UCS when TIMEADT first starts is the first one described in the **example.spec** file. In this case, of course, this is the `american` calendric system. A calendar, property, calendric system, or field value function or table may not be defined or imported more than once in the calendric system specification file.

Next in the example **example.spec** file is a list of the default properties [Soo & Snodgrass 1992]. These property values will be in effect when the TIMEADT system starts. Any property not defined here will have a default value except property `Locale` whose initial value of the empty string. Table 1 lists the properties. Property `override_input_epoch` has the effect in the UCS of accepting the order of epochs defined here for each calendric system.

The formats of period and indeterminate datetime and interval are constructed from determinate datetime and interval, indeterminacy separator, period delimiter and period separator based on the BNF in Figure 3. The end granule should be greater than start granule for indeterminate instant or interval. The grammar for indeterminate instant is ambiguous. The first rule will be applied first.

Following after a list of the default properties are the field value table and field value function declarations. These specify the C structure or function names, and the label used in the calendric system to refer to them.

Next are granularity declarations which specify mapping and anchor between two granularities from different calendars. The anchor will be zero if it is not explicitly specified. Each calendar should have its underlying granularity mapped in terms of other known granularity except the first imported calendar which contains the base line granularity.

Next is renumber statement which changes the external value of a granularity.

Following after the renumber statement are a list of distributions and default values of plausibility, maximum number of iterations during the indeterminate rod counting calculation and semantics. The distribution function takes a double and returns a double. The value of plausibility ranges from 1 (even remotely possible) to 100 (definitely).

The final contend in the specification file are declarations of operating system time function and mapping between operating system time unit and a granularity. These functions provide support for Now and Now-relative in database. The operating system time function takes a poly_int_type as parameter with no return value. The mapping function between granularity and operating system time unit takes two poly_int_type as parameters and returns an integer. The first parameter is number of operating system time units and the second parameter is the number of granules with specified granularity (second in this example). If no error occurs, the function returns 0, otherwise it returns 1.

2.2 Parsing the TIMEADT Specification File

Figures 2 and 4 give the BNF used to parse the **example.spec** file. The details of the specification file are straightforward. Most of the tokens which must be matched are keyword strings. Any line that begins with a hash mark (#) is a comment line and the contents of the rest of that line should be ignored.

Spaces, tabs, and newlines are not significant, except that they separate tokens. Case is significant in the parsing of the file, but as discussed above will not necessarily be significant in the generation of C code to initialize the system. The parsing begins by trying to evaluate the ⟨cs def⟩ token. One or more ⟨import statement⟩ appear first in the file, followed by ⟨rename statement⟩. The order of other top-level tokens does not matter.

The ⟨C string⟩ token is a regular C string [Kernighan & Ritchie 1988] except that it uses single quote instead of double quote. It begins with a single quote, followed by some number of characters and is terminated by a single quote. Single quote can be embedded in the string by '\'. Double quote does not

```
⟨cs def⟩                    ::= {⟨import statement⟩}+ {⟨rename statement⟩}{⟨cs statement⟩ | ⟨property value⟩
                                  | ⟨field value function⟩ | ⟨field value table⟩ | ⟨granularity declaration⟩
                                  | ⟨renumber statement⟩ | ⟨time func mapping⟩ }
⟨import statement⟩          ::= Import ⟨calendar id⟩ from ⟨directory path⟩
⟨rename statement⟩          ::= Rename ⟨calendar id⟩.⟨gran id⟩ as ⟨gran id⟩
⟨cs statement⟩              ::= Define calendric system ⟨cs id⟩
                                 as ⟨calendar epoch list⟩ ⟨calendar order list⟩
⟨calendar epoch list⟩       ::= ⟨calendar epoch list⟩ ‘,’ ⟨calendar epoch entry⟩ | ⟨calendar epoch entry⟩
⟨calendar epoch entry⟩      ::= ⟨calendar id⟩ ⟨C string⟩
⟨calendar order list⟩       ::= with default input order ⟨SQL id list⟩
⟨property value⟩            ::= Property ⟨property name⟩ is ⟨C string⟩
⟨property name⟩             ::= locale | instant_input_format | instant_output_format
                                  | interval_input_format | interval_output_format
                                  | period_input_separator | period_output_separator
                                  | period_output_delimiters | indeterminacy_input_separator
                                  | now_separator | indeterminacy_output_separator
                                  | default_input_distribution | missing_distribution
                                  | distribution_input_format | distribution_output_format
                                  | override_input_epoch | beginning | forever | now
⟨cs property⟩               ::= ⟨calendar epoch⟩ | ⟨epoch input format⟩
⟨calendar epoch⟩            ::= Calendar epoch is ⟨calendar id⟩
⟨field value function⟩      ::= Field value table ⟨C id⟩ is ⟨field value name⟩
                                  from ⟨integer⟩ to ⟨integer⟩
⟨field value table⟩         ::= Field value function ⟨C id⟩ is ⟨field value name⟩
                                  from ⟨integer⟩ to ⟨integer⟩
⟨epoch input format⟩        ::= Input format is ⟨C string⟩
⟨granularity declaration⟩   ::= Granularity ⟨gran id⟩ is ⟨integer⟩ ⟨gran id⟩ [ with anchor ⟨integer⟩
                                    ⟨gran id⟩ ]
                                  | Granularity ⟨gran id⟩ is irregular ⟨gran id⟩ with
                                    ⟨func declaration⟩
⟨func declaration⟩          ::= functions ⟨SQL id⟩ ‘,’ ⟨SQL id⟩ ‘,’ ⟨SQL id⟩
⟨renumber statement⟩        ::= Renumber granularity ⟨gran id⟩ as ⟨integer⟩
⟨time func mapping⟩         ::= ⟨time func⟩ ⟨time mapping⟩
⟨time func⟩                 ::= Operating system time function is ⟨SQL id⟩
⟨time mapping⟩              ::= Granularity ⟨gran id⟩ is ⟨integer⟩ operating system time unit
                                    [with anchor ⟨integer⟩ ]
                                  | Granularity ⟨gran id⟩ is irregular operating system time
                                    unit with function ⟨SQL id⟩
⟨calendar id⟩               ::= ⟨SQL id⟩
⟨gran id⟩                   ::= ⟨SQL id⟩
⟨SQL id list⟩               ::= ⟨SQL id list⟩ ‘,’ ⟨calendar id⟩ | ⟨calendar id⟩
```


Figure 2: Calendric System BNF

⟨determinate instant⟩ ::= ⟨instant_input_format⟩ | ⟨beginning⟩ | ⟨forever⟩ | ⟨now⟩ | ⟨now-relative⟩
⟨now-relative⟩ ::= ⟨now⟩ ⟨now_seperator⟩ ⟨determinate interval⟩
⟨indeterminate instant⟩ ::= ⟨determinate instant⟩ ⟨indeterminate_input_seperator⟩ ⟨determinate instant⟩
    | ⟨now⟩ ⟨now_seperator⟩ ⟨indeterminate interval⟩
⟨instant⟩ ::= ⟨determinate instant⟩ | ⟨indeterminate instant⟩
⟨period⟩ ::= ⟨left_delimitor⟩ ⟨instant⟩ ⟨period_input_separator⟩ ⟨instant⟩
    ⟨right_delimitor⟩
⟨determinate interval⟩ ::= ⟨interval_input_format⟩
⟨indeterminate interval⟩ ::= ⟨determinate interval⟩ ⟨indeterminate_input_seperator⟩
    ⟨determinate interval⟩
⟨interval⟩ ::= ⟨determinate interval⟩ | ⟨indeterminate interval⟩
⟨left_delimiter⟩ ::= ‘(’ | ‘[’
⟨right_delimiter⟩ ::= ‘)’ | ‘]’

Figure 3: Temporal Instant Format BNF

⟨field value name⟩ ::= ⟨character set⟩_⟨language⟩_⟨field name⟩
⟨character set⟩ ::= *ascii* | *gb* | *hz* | *jis* | *unicode* | *latex* ...
⟨language⟩ ::= ⟨written language⟩ | ⟨romanized spoken language⟩
⟨written language⟩ ::= *arabic* | *chinese* | *danish* | *english* | *japanese* | *russian* ...
⟨romanized spoken language⟩ ::= *romanized_mandarin* | *romanized_cantonese*
    | *romanized_japanese* | *romanized_russian* ...
⟨field name⟩ ::= ⟨calendar field name⟩ | *ordinal_name* | *cardinal_symbol*
    | *ordinal_symbol* | *cardinal_name*
⟨calendar field name⟩ ::= ⟨calendar⟩_⟨field⟩_*names*
⟨calendar⟩ ::= *gregorian* | *time_card* | *magnitude* | *ua_academic* ...
⟨field⟩ ::= *day_of_week* | *month* | *meridiem* ...

Figure 4: Field Value Names BNF

Property	*Description*
`Beginning`	`Name for an event preceding any other`
`Default input distribution`	`Default distribution name`
`Distribution input format`	`Input format string for temporal instant with distribution`
`Distribution output format`	`Output format string for indeterminate temporal instant`
`Forever`	`Name for a event following any other`
`Indeterminacy input separator`	`Input separator for indeterminate instant interval`
`Indeterminacy output separator`	`Output separator for indeterminate instant or interval`
`Instant input format`	`Input format string for datetime`
`Instant output format`	`Output format string for datetime`
`Interval input format`	`Input format string for interval`
`Interval output format`	`Output format string for interval`
`Locale`	`Location for timezone displacement`
`Missing distribution`	`Missing distribution name`
`Now`	`Input format string for temporal instant NOW`
`Now separator`	`Input separator for temporal instant NOW-relative`
`Override input epoch`	`Epoch to use first for constant translation`
`Period input separator`	`Input separator for period`
`Period output separator`	`Output separator for period`
`Period output delimiters`	`Output delimiters for period`

Table 1: Properties

need to be escaped. A ⟨C id⟩ is a legal C identifier [Kernighan & Ritchie 1988]. The ⟨SQL id⟩ represents a SQL identifier. A valid SQL identifier is, roughly, a string of not more than eighteen characters, which begins with a letter, and is followed by a combination of letters, numbers and underscores [Date 1987].

A ⟨directory path⟩ is a directory path on the implementation machine. On a Unix computer, the directory path would include any specifiable file path.

Any error in the parsing or processing of this file should terminate processing. There are limits on what tokens evaluate to strings or C identifiers, as the semantic checking phase of parsing of the **example.spec** file must determine. The ⟨calendar id⟩ tokens may only be selected from the list of imported calendars or the special empty calendar, `none`. The event constant specified by C String in calendar epoch entry clause must be understandable according to the epoch calendar and epoch input format.

2.3 Field Value Names, Table and Functions in TIMEADT

The ⟨field value name⟩ non-terminals (given in Figure 4) are not complete. This BNF is intended to be a representative sampling of different languages that may be supported by TIMEADT. This list of different languages, character sets, and field names is simply a starting point. Each of the different terminals must be SQL identifiers.

Below we give the first few elements of several different field value tables and functions as examples. Notice that the names follow the form of ⟨character set⟩_⟨language⟩_⟨field name⟩

- `ascii_english_gregorian_day_of_week_names` = { `Monday, Tuesday, ...` }

- `latex_german_gregorian_month_names` = { `Januar, Februar, Mi\"{a}rz` (Miärz), ... }
- `ascii_romanized_japanese_gregorian_day_of_week_names` = { `getsuyoobi, kayoobi, suiyoobi, ...` }
- `jis_japanese_gregorian_day_of_week_names` = { *JIS representation* }
- `unicode_chinese_chinese_lunar_month_names` = { *unicode representation* }
- `ascii_romanized_russing_gregorian_month_names` = { `Jinvar, ...` }
- `hz_chinese_chinese_lunar_month_names` = { `R;TB, 6~TB, ...` }
- `ascii_arabic_cardinal_symbols` = { `1, 2, 3, ...` }
- `ascii_english_ordinal_symbols` = { `1st, 2nd, 3rd, ...` }
- `ascii_romanized_mandarin_cardinal_names` = { `yi, er, san, si, ...` }

3 The Calendar Specification Files

Each calendar is described by a specification file, the contents of which are provided by the calendar implementor. The specification file should not be changed by the database administrator. The database administrator should move the calendar's specification file and object file to some location and place the "path" to the calendar in the TIMEADT specification file.

3.1 Example Calendar Specification File

Figures 5 and 6 are example specification files for, respectively, the `Gregorian` and `Astronomy` calendars.

Every calendar has a *long name*, which is prepended to many C identifiers associated with that calendar, such as function names and certain structures. The long name is required to be a legal C identifier. The name given in the calendric system specification file, called the *short name* is used to refer to the calendar in the UCS and at the SQL level and it must be a SQL identifier. The other contents of the calendar specification file are the list of special functions in the calendar and the list of field names used.

Calendar implementors should use a lengthy and specific long name, since this is the only distinguishing difference between two calendars. We suggest adding the version to the calendar long name. The TIMEADT system may not contain two calendars with the same long name. The long name is used to refer to the calendar, and also as the middle part of function names for calendar regular functions which must be implemented for each calendar (see Table 2).

After the long name, the next component of a calendar specification file is underlying granularity declaration and granularity declarations. The BNF for granularity declaration is shown in Figures 2. The order of declarations is important because an integer number from 0 to $n-1$ (n is the total number of granularities in the calendar) will be assigned to each granularity as the local id according to this order.

Next in calendar specification file are additional field names used by a calendar. The field names consist of granularity names and additional field names. They are used to communicate field values when parsing and generating temporal constants. The order of additional field names is important. Base on this order, each additional field name will be assigned an integer number starting from n (n is the total number of granularities in the calendar). This number is the index to the array of field values.

```
#Standard University of Arizona American Gregorian Calendar version 1
#file: gregorian.spec

Calendar is standard_gregorian_v1

Granularity second is underlying granularity

Granularity minute is irregular second with functions
    greg_cast_minute_to_second, greg_cast_second_to_minute,
    greg_scale_minute_to_second

Granularity hour is  60 minute with anchor 0 minute

Granularity day is 24 hour with anchor 0 hour

Granularity week is 7 day with anchor 0 day

Granularity month is irregular day with functions
    greg_cast_month_to_day, greg_cast_day_to_month,
    greg_scale_month_to_day

Granularity year is irregular day with functions
    greg_cast_year_to_day, greg_cast_day_to_year,
    greg_scale_year_to_day

Additional field names are  era,  century, year_of_century,
  decade, year_of_decade, day_of_week, day_of_year,
  meridiem, milli, micro, nano,
  era2, year2, century2, year_of_century2, decade2, year_of_decade2,
  month2, day2, day_of_week2, day_of_year2,
  meridiem2, hour2, minute2, second2, milli2, micro2, nano2,
  month_interval1, month_interval2,
  1st_start, 1st_end, 2nd_start, 2nd_end, open_period,
  closed_period
```

Figure 5: Gregorian Calendar Specification file (**gregorian.spec**)

```
#Astronomy calendar

Calendar is standard_astronomy_v1

Granularity second is underlying granularity

Granularity day_hundredth is 864 second with anchor 0 second

Granularity day is 100 day_hundredth with anchor 0 day_hundredth

Granularity year is irregular day with functions
  astro_cast_year_to_day, astro_cast_day_to_year,
  astro_scale_det_year_to_day

Granularity century is 100 year with anchor 0 year

Additional field names are day2, year2, century2
```

Figure 6: Astronomy Calendar Specification File (**astronomy.spec**)

$\langle$calendar description$\rangle$	::=	$\langle$calendar name$\rangle$ $\{$ $\langle$underlying gran decl$\rangle$ $\mid$ $\langle$granularity declaration$\rangle$ $\}$ $\langle$field names$\rangle$
$\langle$calendar name$\rangle$	::=	*`Calendar is`* $\langle$calendar id$\rangle$
$\langle$underlying gran decl$\rangle$	::=	*`Granularity`* $\langle$gran id$\rangle$ *`is underlying granularity`* *`[ with local ID`* $\langle$integer$\rangle$ *`]`*
$\langle$field names$\rangle$	::=	*`Additional field names are`* $\langle$field name list$\rangle$ $\mid$ ϵ
$\langle$field name list$\rangle$	::=	$\langle$field name list$\rangle$ ‘`,`’ $\langle$field name$\rangle$ $\mid$ $\langle$field name$\rangle$
$\langle$field name$\rangle$	::=	$\langle$SQL id$\rangle$

Figure 7: BNF Used to Describe Calendar Specification Files

3.2 Parsing the Specification File

Figure 7 is the BNF used to process the calendar.

The parsing of a calendar specification file is similar to the **example.spec** file. First, any lines which begin with a hash mark are comment lines and are ignored. Spaces and newlines are not significant except that they separate tokens. The parsing begins by trying to evaluate the ⟨calendar description⟩ token.

The calendar's long name must be unique across all calendars defined in this TIMEADT program. The token ⟨field name⟩ is a list of the valid field names.

4 List of Errors

Any error in the parsing TIMEADT specification file or calendar specification file will terminate the generation program. The error message of syntactic error has the following format: ⟨filename⟩: error in line ⟨n⟩ before ⟨word⟩. The semantic errors are listed below.

- The mappings with anchor specification among calendars are not exactly n-1 when there are n calendars.

  Error message: <filename>: each calendar should have its underlying granularity mapped in terms of other known granularity exactly once except <cal_name> calendar.

- The referenced granularity in granularity declaration or additional mapping is not declared.

  Error message: <filename> error in line <n> before <word>: granularity not declared.

- Granularity declared more than once.

  Error message: <filename> error in line <n> before <word>: granularity declared more than once.

- Granularity does not exist in rename statement or renumber statement.

  Error message: <filename> error in line <n> before <word>: granularity not declared.

- Calendar does not exist.

  Error message: <filename> error in line <n> before <word>: calendar not found.

- Specified number is out of range in renumber statement.

  Error message: <filename> error in line <n>: <num> exceeds the total number of granularities.

5 Calendar Specifics

Calendars need to provide a certain interface so they may be integrated more easily into the system. Of course, a calendar must provide a specification file as discussed above. Each calendar must provide two standard calendar functions (Table 2, in Section A).

Each calendar must also provide an initialization function, the name of which is the combination of "cal_" plus the calendar's long name (as given in the specification file) followed by _init(int calendar_number). This function takes a single parameter, the calendar's identification number. This initialization function should allocate any data structures used internally by the calendar's functions. For example, in the Gregorian calendar the initialization function would be declared as cal_standard_gregorian_calendar_v1_init(int cal_number). This function is called from the function init_calendars() which is called after the UCS is initialized.

6 Summary

We described the contents and construction of the calendar and calendric system specification files. We discussed the processing of these files, and how to generate C files which integrate the calendars into the TIMEADT system.

The system described here is comprehensive yet simple, and facilitates easy addition of calendars and calendric systems.

Acknowledgments

Thanks to Curtis Dyreson and Mike Soo for comments on earlier drafts of this paper. Thanks also to Leo So for comments and for his work on the format and description of field table entries, and for providing the BNF descriptions.

Bibliography

[Date 1987] Date, C. J. "A Guide to the SQL Standard." Addison-Wesley, 1987.

[Kernighan & Ritchie 1988] Kernighan, B.W. and D.M. Ritchie. "The C Programming Language." Vol. second edition of Prentic Hall Software Series. Englewood Cliffs, NJ: Prentice Hall, 1988.

[Kline 1993] Kline, N. "The Gregorian Calendar Specification." TempIS Technical Report 47. Computer Science Department, University of Arizona. April 1993.

[Soo & Snodgrass 1992] Soo, M. D. and R. Snodgrass. "Mixed Calendar Query Language Support for Temporal Constants." TempIS Technical Report 29. Computer Science Department, University of Arizona. Revised May 1992, 59 pages.

A The Generated C Files

This section contains the C files generated by the descriptions in the above sections, named **example.c** and **example.h**. **example.c** is given in Section A.2 and **example.h** is given in Section A.4. **example.c** contains three main functions, `init_calendars()`, `init_properties()` and `init_gran_graph()`. There are also accompanying data structures, and a section with data structures for the field value tables and functions. At system initialization time, `timeadt_init()` is called and it in turn calls four initialization functions `init_distribution()`, `init_gran_graph()`, `ucs_init()` and `init_event_constants()`. `init_distribution()` initialize the distributions declared in TIMEADT specification file. `init_gran_graph` will construct the granularity graph. Within `ucs_init()` routine, , `init_gen()` is called, which will call the property routine initialization routine (`init_property()`), and the calendar and calendric system initialization function (`init_calendars()`). `init_event_constants()` initialize the event constants of epochs. The generated C file is ANSI C compatible [Kernighan & Ritchie 1988].

A.1 The Contents of example.c

This section and the next few describe the contents of the generated C file **example.c**.

At the top of this file is a comment giving the time the file was created and the last date of change of the calendar and calendric system specification files. Throughout this file are comments that explain its contents.

A.1.1 Calendar and Calendric System Generation

After the initial contents are several constants which are used to communicate table sizes to the rest of TIMEADT. `NumCalendars` and `NumberOfCS` describe the number of calendars and calendric systems specified in the **example.spec** file. The next two constants, `fv_max_field_functions` and `fv_max_field_tables` give the number of field value functions and tables, respectively supplied by the **example.spec** file. The maximum number of arguments which may be supplied to functions is defined by `gen_fv_max_formal_args`. The variable `ucs_cal_number` is an internal variable used by the UCS which specifies the current calendar number.

There are several variables and tables required for the declaration of each calendar, with similar information for each calendric system.

The example **example.spec** presented earlier will be used as the basis of the generated C file. As Figure 1 shows, that example used two calendars, the Gregorian and Astronomy calendars. The declarations for the Gregorian calendar come first, since it was imported first. A function reference to the Gregorian calendar's initialization routine is given. Notice that this takes one parameter which is the calendar's identification number. Next are function references to the calendar's functions, each of which is preceded by a comment which gives the SQL name used in function binding to identify that function. Following this is a sequence of structures which describe the return types of the calendars.

Next comes the field value names. This is a character array containing the field names the calendar and UCS will use to communicate with each other. This list must be in the same order that the information was specified in the calendar's specification file.

This sequence of structures is repeated for each calendar, based on the information in that calendar's specification file. The calendric system structures follow after the calendar structures.

The `calendar_list` variables describes the information in each calendar. These are separated once again to facilitate use by C's static array initialization rules. The Gregorian calendar is calendar number `1`, has `0` functions, has a pointer to the function structures, has 21 field names, and has a pointer to the field name list. Each calendar also has a field name list used to communicate field values between a calendar and the UCS.

This list is allocated in the routine `init_calendars`. Similar information is provided for the Astronomy calendar. Next is the `cal_epochs_event` structure, an array of `seconds_type` which gives the start and stop events for each epoch within a calendric system. These are in the format of the `seconds structure` [Soo & Snodgrass 1992]. The end of one epoch is one chronon before the start of the next. After this comes the `cal_epochs_calendars` list, which defines which calendar is to be used during each epoch defined in the `cal_epochs_events` structure. Finally, we have the list of calendric systems (`calendric_system_list`).

Following this information is the routine `init_calendars()` which does the actual calendar initialization and combines the structures, filling in information which may not be encoded staticly in them.

First it calls the calendar's initialization functions. Then, for each calendric system, it creates an epoch structure. Using the entries from the `cal_epochs_events` table it creates intervals for each calendar.

A.1.2 Property Generation

The list of property names follow the calendar structures. Next comes the property structures themselves, and these are declared just above the property initialization routine. The declarations begin with two variables which are used to maintain the property list. Next follows the list of initial property values, in the order defined by the `property_name_labels` defined in the file **h/ucs/property.h**.

The rest of the property declarations are boiler plate and do not change. Only the string array `ucs_char 0_property_init_val` structure changes.

A.1.3 Field Value Generation

Next in the **example.c** file is the field value initialization. Static arrays describing both field value functions and field value tables are all that is needed. These are used at run time to look up functions or tables.

A.1.4 Multiplexing Between the UCS and Calendars

The UCS must multiplex between the different calendar systems when it calls *regular* calendar functions (termed this because they are required to be implemented by all calendars). The code to accomplish this follows next. As a motivating example, consider a string with datetime format that needs to be converted to an temporal instant. The UCS first finds an appropriate calendar based on the current datetime input format. After parsing the string and marking the array of field values, the UCS asks that calendar to convert the array of field values to an poly_int_type and a granularity by calling *regular* calendar function `fvt_to_poly`. The UCS needs a table of all regular functions to do this.

The UCS uses a two dimensional table of function pointers, with the first dimension the number of special calendar functions (2) and the second the number of calendars (2 in the case of Figure 1).

Table 2 is a list of the regular calendar functions, broken down into two categories.

Event and Interval Translation Functions
`error_type poly_to_fvt(poly_int_type, granularity_type, int, value_array_type*)`

Constant Translation Functions
`error_type fvt_to_poly(value_array_type*, poly_int_type, granularity_type*, int*)`

Table 2: Regular Calendar Function Table

The array contains function pointers to each of the regular calendar functions. The function declarations are formed by taking each function name from the table and replacing `calendar` with the long name of each of the defined calendars. For example, since we have two calendars, there are two versions of the function `fvt_to_poly`:

`cal_standard_gregorian_v1_fvt_to_poly`
`cal_standard_astronomy_v1_fvt_to_poly`

A.1.5 Mappings between local id and global id of a granularity

Next in the **example.c** file are mappings between local id and global id of a granularity. The global id is also the external value of a granularity used by users. The mappings consist of several arrays and a function called `get_global_id`. `local_id_map` is used to find the local id of a granularity given its global id. If the local id of a granularity and the name of calendar it belongs to are given, function `get_global_id` will return the global id of the granularity.

A.1.6 Constructing granularity graph

The final part of the **example.c** file is the construction of granularity graph. After initializing the lattice, each granularity is declared exactly once by calling function `declare_granularity_with_anchor` or `declare_granularity` except the base line granularity which does not need to be declared. Next are the mapping declarations. Mappings can be regular, irregular, or congruent. Each granularity should have at least one mappings except base line granularity. Finally, function `declare_done` is called and if no error occurs, then granularity graph is successfully constructed. Whenever an error occurs, the program will print an error message and exit right away.

A.2 Example C Source File example.c

The following is the generated C source file from the examples presented in this document.

```
/*********************************************************************\
*                                                                     *
*   File: example.c
*                                                                     *
*   The TimeADT System is free software in the public domain; you can   *
*   redistribute it and/or modify it as you wish. We ask that you       *
*   retain credits referencing the University of Arizona and that you   *
*   identify any changes you make.                                      *
*                                                                     *
*   Report problems to rts@cs.arizona.edu                               *
*   Direct all inquiries to:    The TimeADT Project                     *
*                               Department of Computer Science          *
*                               University of Arizona                   *
*                               Tucson, AZ 85721                        *
*                               U.S.A.                                  *
*                                                                     *
\*********************************************************************/
/*
 * example.c:
 *  This file contains code which is automatically generated by the
 *  TimeADT generation facility.
 *
 * It was generated on Fri May 28 00:05:31 1999
 *
 * from file example.spec, last changed Wed May 26 21:21:18 1999
 *
 * ./gregorian.spec, last changed Thu May 20 22:12:58 1999
```

```
 * ./astronomy.spec, last changed Thu May 20 22:12:58 1999
 *
 * This file also contains the code to initialize the properties and
 * field value functions and tables.
 *
 * Initialization routine: timeadt_init() do the following
 *                         initialize granularity graph
 *                         initialize UCS
 *                         initialize event constants in the epochs
 *
 * Externally, call init_gen(), which calls init_calendars()
 *  and init_properties.
 */
#include <stdio.h>
#include "system_wide.h"
#include "fv.h"
#include "ucs.h"
#include "fv.h"
#include "poly_int.h"
#include "gran.h"
#include "PMF.h"
#include "now.h"
#include "example.h"
/*
 *  The following are constants used to communicate the size of generated tables
 */

int NumOfGran = 15;
int NumCalendars = 2;
int NumberOfCS = 1;
int gen_fv_max_field_functions = 1;
int gen_fv_max_field_tables = 3;
unsigned char ucs_cal_number;

/* semantics default value */
unsigned char plausability = 50;
unsigned int max_iters= 15;
int semantics_finest = 1;
int semantics_coarsest = 0;
int semantics_lhs = 0;
int semantics_rhs = 0;
int semantics_cast = 1;

/*
 * Routine:  init_gen
 *
 * Description: calls the initialization routines for calendars and properties
 *
 * Arguments: None
 *
 * Return Value: Error code
 *
 * Errors: None
 *
 * Side Effects: None
 */


void init_gen()
```

```
{
  init_calendars();
  init_properties();
}

/* calendar and calendric system initialization */

void standard_gregorian_v1_init(int);

char  *standard_gregorian_v1_fn_list[] = {
  "nanosecond",   "microsecond",   "millisecond",
  "second",   "minute",   "hour",
  "day",   "week",   "month",
  "year",   "decade",  "era",
  "century",  "year_of_century",  "year_of_decade",
  "month_of_year",  "day_of_year",  "day_of_month",
  "day_of_week",  "hour_of_day",  "minute_of_hour",
  "second_of_minute",  "millisecond_of_second",  "microsecond_of_millisecond",
  "nanosecond_of_microsecond",};

void standard_astronomy_v1_init(int);

char  *standard_astronomy_v1_fn_list[] = {
  "second",   "day_hundredth",   "day",
  "year",   "century", };

calendar_type calendar_list_0 = {
       "Gregorian",
       1,
       21,
       standard_gregorian_v1_fn_list,
       NULL
};

calendar_type calendar_list_1 = {
       "Astronomy",
       2,
       6,
       standard_astronomy_v1_fn_list,
       NULL
};

calendar_type *calendar_array[] =  {
      &calendar_list_0,
      &calendar_list_1,
};

char* cal_epochs_events[][2] = {
   {
      "beginning",
      "September 14, 1752", },
};

calendar_type *cal_epochs_calendars[] = {
&calendar_list_1, &calendar_list_0};

calendric_system_type calendric_system_list[] ={
   {  "american",
      2,
```

```
      "Gregorian",
      "<month_of_year,english_month_names> <day_of_month,arabic_numeral>,
<year,arabic_numeral>",
      NULL
    }};

void init_calendars()
{
  int i, j;

  int cnt; /* used to walk through system */
  epoch_type *ep;

  calendar_list_0.field_values = (value_array_type*)
  ucs_malloc(sizeof(value_array_type)*calendar_list_0.num_field_names);
  for (i=0;i<calendar_list_0.num_field_names;i++)
    calendar_list_0.field_values[i].value =
      (actual_arg_type)ucs_malloc(sizeof(actual_arg_type));

  calendar_list_1.field_values = (value_array_type*)
  ucs_malloc(sizeof(value_array_type)*calendar_list_1.num_field_names);
  for (i=0;i<calendar_list_1.num_field_names;i++)
    calendar_list_1.field_values[i].value =
      (actual_arg_type)ucs_malloc(sizeof(actual_arg_type));

  standard_gregorian_v1_init(1);
  standard_astronomy_v1_init(2);
  cnt = 0;


  for (i=0; i<NumberOfCS; i++) {
    ep = ucs_allocate_epoch_type(calendric_system_list[i].num_epochs);

    for(j=0; j<calendric_system_list[i].num_epochs; j++) {
      ep[j].start = cal_epochs_events[i][j];
      ep[j].calendar = cal_epochs_calendars[cnt];
      cnt++;
    }
    calendric_system_list[i].epochs = ep;
  }

  ucs_declare_local_calendric_system(calendric_system_list[0].name);
  ucs_declare_global_calendric_system(calendric_system_list[0].name);
}

boolean ucs_non_activated_props=FALSE;
prop_node *initial_property_list[NUM_PROPERTIES];
prop_marker *ucs_front_properties, *ucs_end_properties;

/* property names, in order*/

char ucs_locale[] = "Locale";
char ucs_instant_input_format[] = "Instant Input Format";
char ucs_instant_output_format[] = "Instant Output Format";
char ucs_interval_input_format[] = "Interval Input Format";
char ucs_interval_output_format[] = "Interval Output Format";
char ucs_now_separator[] = "Now Separator";
char ucs_period_input_separator[] = "Period Input Separator";
char ucs_period_output_separator[] = "Period Output Separator";
```

```
char ucs_period_output_delimiters[] = "Period Output Delimiters";
char ucs_indeterminacy_input_separator[] = "Indeterminacy Input Separator";
char ucs_indeterminacy_output_separator[] = "Indeterminacy Output Separator";
char ucs_default_input_distribution[] = "Default Input Distribution";
char ucs_missing_distribution[] = "Missing Distribution";
char ucs_distribution_input_format[] = "Distribution Input Format";
char ucs_distribution_output_format[] = "Distribution Output Format";
char ucs_override_input_epoch[] = "Override Input Epoch";
char ucs_beginning[] = "Beginning";
char ucs_forever[] = "Forever";
char ucs_now[] = "Now";

char *ucs_char_property_name[] = {
    ucs_locale,
    ucs_instant_input_format,
    ucs_instant_output_format,
    ucs_interval_input_format,
    ucs_interval_output_format,
    ucs_now_separator,
    ucs_period_input_separator,
    ucs_period_output_separator,
    ucs_period_output_delimiters,
    ucs_indeterminacy_input_separator,
    ucs_indeterminacy_output_separator,
    ucs_default_input_distribution,
    ucs_missing_distribution,
    ucs_distribution_input_format,
    ucs_distribution_output_format,
    ucs_override_input_epoch,
    ucs_beginning,
    ucs_forever,
    ucs_now,
};

/* Initial property values, in same order as above */
char *ucs_char_property_init_val[] = {
    "Tucson",
    "<month_of_year,english_month_names> <day_of_month,arabic_numeral>,
<year,arabic_numeral>",
    "<month_of_year,english_month_names> <day_of_month,arabic_numeral>,
<year,arabic_numeral>",
    "<month,arabic_numeral> months",
    "<month,arabic_numeral> months",
    " ",
    " - ",
    " - ",
    "[)",
    " ~ ",
    " ~ ",
    "uniform",
    "missing",
    "<value> with <distribution> distribution",
    "<value> with <distribution> distribution",
    "Gregorian",
    "beginning",
    "forever",
    "now",
};
```

```
/* init properties to their initial value */

void init_properties() {
  int i;
  /* init empty prop list to NULL */
  ucs_front_properties = ucs_end_properties =
      (prop_marker*)ucs_malloc(sizeof(prop_marker));
  ucs_end_properties->next = ucs_end_properties->prev = NULL;

  for(i=0;i<NUM_PROPERTIES;i++) {
    initial_property_list[i] =(prop_node*)ucs_malloc(sizeof(prop_node));
    initial_property_list[i]->prop.num = i;
    initial_property_list[i]->prop.val = ucs_char_property_init_val[i];
  }

  for(i=0;i<NUM_PROPERTIES-1;i++)
    initial_property_list[i]->next = initial_property_list[i+1];

  initial_property_list[i]->next = NULL;

  ucs_end_properties->prop_list = initial_property_list[0];

}

char *gen_function_names_alias_names[]={
    "arabic_numeral",
    "english_month_names",
    "mandarin_month_names",
    "danish_month_names",
};

char *gen_function_names_mapped_names[]={
    "ascii_arabic_cardinal_symbols",
    "ascii_english_gregorian_month_names",
    "ascii_romanized_mandarin_gregorian_month_names",
    "latex_danish_gregorian_month_names",
};

/* calendar: standard_gregorian_v1 */

error_type cal_standard_gregorian_v1_poly_to_fvt(void*a);
error_type cal_standard_gregorian_v1_fvt_to_poly(void*a);


/* calendar: standard_astronomy_v1 */

error_type cal_standard_astronomy_v1_poly_to_fvt(void*a);
error_type cal_standard_astronomy_v1_fvt_to_poly(void*a);


function_ptr cal_table[num_cal_functions*num_calendars] = {

/* calendar: standard_gregorian_v1 */
  {cal_standard_gregorian_v1_poly_to_fvt},
  {cal_standard_gregorian_v1_fvt_to_poly},

/* calendar: standard_astronomy_v1 */
  {cal_standard_astronomy_v1_poly_to_fvt},
  {cal_standard_astronomy_v1_fvt_to_poly}
```

```
};

void init_gran_graph();
void init_distribution();

/* Routine:      timeadt_init
 *
 * Descriptioin: the initialization routine for TimeADT.
 *
 * Arguments:    None
 *
 * Return value: None
 *
 * Error:   If any error occurs in init_gran_graph or init_event_constants,
 *          the program will print out error message and halt.
 *
 * Side Effects: None
 */

void timeadt_init()
{
    init_distribution();
    init_gran_graph();
    ucs_init();
    init_event_constants();
}


/* Routine:      init_distribution
 *
 * Descriptioin: initialize the distribution functions and PMFTrees.
 *
 * Arguments:    None
 *
 * Return value: None
 *
 * Error:        None
 *
 * Side Effects: None
 */

void init_distribution()
{
    int i;

    PMF_dists.num = 2;
    for ( i= 0; i< PMF_dists.num ; i++)
        PMF_dists.dists[i] = NULL;

    PMF_dists.funcs[0] = PMF_uniform;
    PMF_dists.names[0] = strdup("uniform");
    PMF_dists.funcs[1] = PMF_pyramid;
    PMF_dists.names[1] = strdup("pyramid");

    dist_in.format[0] =0;
    dist_out.format[0] = 0;
}

/* The total number of granularities in granularity graph */
```

```
int max_num_gran = 15;

/* The mapping between local granualrity id and global granulairty id */

int local_id_map[] = {0, 1, 2, 3, 4, 5, 7, 8, 9, 0, 1, 6, 2, 3, 4};
static int cal_number = 2;
static char *cal_names[]= { "Gregorian", "Astronomy"};
static int global_id_map[][24] = { {0, 1, 2, 3, 4, 5, 11, 6, 7, 8}, {9, 10, 12, 13, 14}};

/* Internal function: mapping local id to global id
 * Auguments: calendar name and local id
 * Return value: global id
 */
unsigned char get_global_id(char *cal_name, int local_id)
{
    int i;
    for ( i =0; i< cal_number; i++) {
        if ( strcasecmp(cal_name, cal_names[i]) == 0)
            break;
    }

    return global_id_map[i][local_id];
}


/* calendar: standard_gregorian_v1 */

extern gran_error_type greg_cast_minute_to_second(int, int, poly_int_type, poly_int_type);
extern gran_error_type greg_cast_second_to_minute(int, int, poly_int_type, poly_int_type);
extern gran_error_type greg_scale_minute_to_second(int, int, poly_int_type, poly_int_type,
                                                   poly_int_type);
extern gran_error_type greg_cast_month_to_day(int, int, poly_int_type, poly_int_type);
extern gran_error_type greg_cast_day_to_month(int, int, poly_int_type, poly_int_type);
extern gran_error_type greg_scale_month_to_day(int, int, poly_int_type, poly_int_type,
                                               poly_int_type);
extern gran_error_type greg_cast_year_to_day(int, int, poly_int_type, poly_int_type);
extern gran_error_type greg_cast_day_to_year(int, int, poly_int_type, poly_int_type);
extern gran_error_type greg_scale_year_to_day(int, int, poly_int_type, poly_int_type,
                                              poly_int_type);

/* calendar: standard_astronomy_v1 */

extern gran_error_type astro_cast_year_to_day(int, int, poly_int_type, poly_int_type);
extern gran_error_type astro_cast_day_to_year(int, int, poly_int_type, poly_int_type);
extern gran_error_type astro_scale_year_to_day(int, int, poly_int_type, poly_int_type,
poly_int_type);


/*
 * Routine:      init_gran_graph
 *
 * Description:  Constructs granularity graph
 *
 * Arguments:    None
 *
 * Return value: None
 *
 * Error:        Results from error in initializing lattice, declaring
```

```
 *              granularity or declaring mapping. If error occurs,
 *              exit right away.
 *
 * Side Effects: None
 */

void init_gran_graph()
{
    poly_int_type anchor;

    if ( init_lattice() != gran_OK) {
        fprintf(stderr, "Failed to create LATTICE\n");
        exit(1);
    }

    poly_from_int(anchor, 1, 0);
    if ( declare_granularity_with_anchor(astro_second, 1,  anchor, second) !=
         gran_OK){
        fprintf(stderr, "Error: declare gran astro_second.\n");
        exit(1);
    }

    /* declare granularity for Gregorian calendar */
    poly_from_int(anchor, 1, 0);
    if ( declare_granularity_with_anchor(microsecond, 1, anchor, nanosecond) !=
         gran_OK){
        fprintf(stderr, "Error: declare gran microsecond.\n");
        exit(1);
    }

    poly_from_int(anchor, 1, 0);
    if ( declare_granularity_with_anchor(millisecond,  2, anchor, microsecond) != gran_OK){
        fprintf(stderr, "Error: declare gran millisecond.\n");
        exit(1);
    }

    poly_from_int(anchor, 1, 0);
    if ( declare_granularity_with_anchor(second, 3, anchor, millisecond) != gran_OK){
        fprintf(stderr, "Error: declare gran second.\n");
        exit(1);
    }

    if ( declare_granularity(minute, 4) != gran_OK){
        fprintf(stderr, "Error: declare gran minute.\n");
        exit(1);
    }

    poly_from_int(anchor, 1, 0);
    if ( declare_granularity_with_anchor(hour, 5, anchor, minute) != gran_OK){
        fprintf(stderr, "Error: declare gran hour.\n");
        exit(1);
    }

    poly_from_int(anchor, 1, 0);
    if ( declare_granularity_with_anchor(day, 6, anchor, hour) != gran_OK){
        fprintf(stderr, "Error: declare gran day.\n");
        exit(1);
    }
```

```
poly_from_int(anchor, 1, 0);
if ( declare_granularity_with_anchor(week, 7, anchor, day) != gran_OK){
    fprintf(stderr, "Error: declare gran week.\n");
    exit(1);
}

if ( declare_granularity(month, 8) != gran_OK){
    fprintf(stderr, "Error: declare gran month.\n");
    exit(1);
}

if ( declare_granularity(year, 9) != gran_OK){
    fprintf(stderr, "Error: declare gran year.\n");
    exit(1);
}

/* declare granularity for Astronomy calendar */
poly_from_int(anchor, 1, 0);
if ( declare_granularity_with_anchor(astro_day_hundredth,  1, anchor,
                                     astro_second) != gran_OK){
    fprintf(stderr, "Error: declare gran astro_day_hundredth.\n");
    exit(1);
}

poly_from_int(anchor, 1, 0);
if ( declare_granularity_with_anchor(astro_day, 2, anchor,
                                     astro_day_hundredth) != gran_OK){
    fprintf(stderr, "Error: declare gran astro_day.\n");
    exit(1);
}

if ( declare_granularity(astro_year, 3) != gran_OK){
    fprintf(stderr, "Error: declare gran astro_year.\n");
    exit(1);
}

poly_from_int(anchor, 1, 0);
if ( declare_granularity_with_anchor(astro_century, 4, anchor, astro_year) !=
     gran_OK){
    fprintf(stderr, "Error: declare gran astro_century.\n");
    exit(1);
}

if ( declare_regular_mapping(microsecond, nanosecond, 1000) != gran_OK) {
    fprintf(stderr, "Error: declare regular map from microsecond to nanosecond.\n");
    exit(1);
}

if ( declare_regular_mapping(millisecond, microsecond, 1000) != gran_OK) {
    fprintf(stderr, "Error: declare regular map from millisecond to microsecond.\n");
    exit(1);
}

if ( declare_regular_mapping(second, millisecond, 1000) != gran_OK) {
    fprintf(stderr, "Error: declare regular map from second to millisecond.\n");
    exit(1);
}

if ( declare_irregular_mapping(minute, second,
```

```
                                &greg_cast_minute_to_second,
                                &greg_cast_second_to_minute,
                                &greg_scale_minute_to_second) != gran_OK) {
        fprintf(stderr, "Error: declare irregular map from minute to second.\n");
        exit(1);
    }

    if ( declare_regular_mapping(hour, minute, 60) != gran_OK) {
        fprintf(stderr, "Error: declare regular map from hour to minute.\n");
        exit(1);
    }

    if ( declare_regular_mapping(day, hour, 24) != gran_OK) {
        fprintf(stderr, "Error: declare regular map from day to hour.\n");
        exit(1);
    }

    if ( declare_regular_mapping(week, day, 7) != gran_OK) {
        fprintf(stderr, "Error: declare regular map from week to day.\n");
        exit(1);
    }

    if ( declare_irregular_mapping(month, day,
                                &greg_cast_month_to_day,
                                &greg_cast_day_to_month,
                                &greg_scale_month_to_day) != gran_OK) {
        fprintf(stderr, "Error: declare irregular map from month to day.\n");
        exit(1);
    }

    if ( declare_irregular_mapping(year, day,
                                &greg_cast_year_to_day,
                                &greg_cast_day_to_year,
                                &greg_scale_year_to_day) != gran_OK) {
        fprintf(stderr, "Error: declare irregular map from year to day.\n");
        exit(1);
    }

    if ( declare_regular_mapping(astro_day_hundredth, astro_second, 864) != gran_OK) {
        fprintf(stderr, "Error: declare regular map from astro_day_hundredth to
                astro_second.\n");
        exit(1);
    }

    if ( declare_regular_mapping(astro_day, astro_day_hundredth, 100) != gran_OK) {
        fprintf(stderr, "Error: declare regular map from astro_day to
                astro_day_hundredth.\n");
        exit(1);
    }

    if ( declare_irregular_mapping(astro_year, astro_day,
                                &astro_cast_year_to_day,
                                &astro_cast_day_to_year,
                                &astro_scale_year_to_day) != gran_OK) {
        fprintf(stderr, "Error: declare irregular map from astro_year to astro_day.\n");
        exit(1);
    }

    if ( declare_regular_mapping(astro_century, astro_year, 100) != gran_OK) {
```

```
        fprintf(stderr, "Error: declare regular map from astro_century to
                astro_year.\n");
        exit(1);
    }

    if ( declare_congruent(astro_second, second) != gran_OK) {
        fprintf(stderr, "Error: declare congruent map from astro_second to second.\n");
        exit(1);
    }

    if ( declare_done() != gran_OK) {
         fprintf(stderr, "Error: Declare done.\n");
         exit(1);
     }
}

/* The following functions provide support for now and now-relative*/

extern void sys_time(poly_int_type );
extern int unit_to_sec(poly_int_type, poly_int_type);

void (*sys_time_func)(poly_int_type) = sys_time;
time_map_type time_map = { second, unit_to_sec };
```

A.3 The Contents of example.h

The **example.h** file contains external values of all granularities. It also exports the main initialization routine for TIMEADT system called `timeadt_init` which is defined in **example.c**.

A.4 Example C Header File example.h

The following is the generated C header file from the examples presented in this document.

```
/*********************************************************************\
*                                                                     *
*   File: example.h                                                   *
*                                                                     *
*   The TimeADT System is free software in the public domain; you can *
*   redistribute it and/or modify it as you wish. We ask that you     *
*   retain credits referencing the University of Arizona and that you *
*   identify any changes you make.                                    *
*                                                                     *
*   Report problems to rts@cs.arizona.edu                             *
*   Direct all inquiries to:    The TimeADT Project                   *
*                               Department of Computer Science        *
*                               University of Arizona                 *
*                               Tucson, AZ 85721                      *
*                               U.S.A.                                *
*                                                                     *
\*********************************************************************/

/* External granularity id */

#define nanosecond 0
#define microsecond 1
#define millisecond 2
#define second 3
```

```
#define minute 4
#define hour 5
#define day 11
#define week 6
#define month 7
#define year 8
#define astro_second 9
#define astro_day_hundredth 10
#define astro_day 12
#define astro_year 13
#define astro_century 14

extern  void timeadt_init();
```