THE TIME SERVER

December 7, 1984

Table of Contents

1. Theory

The Time Server provides all of the time facilities for Accent. It can return the time in a number of different formats.

Internally, time is kept as the number of milliseconds since 00:00 a.m. (midnight), November 17, 1858, Greenwich Mean Time (the Smithsonian time standard). In order to store the time efficiently, it is kept as an ordered pair, the first part representing the number of weeks that have passed since the base date-time and the second part representing the number of milliseconds that have passed in the current week (a week starts at 00:00 a.m. Wednesday). This format also allows efficient comparison of times. Conversions to local time are done using a time zone index and information regarding whether or not to apply daylight savings time.

The zone record, Zone_Info, allows you to specify and receive zone information. Both the zone number and application of daylight savings time may be either specified explicitly or defaulted, depending on the settings of the UseTimeZone and UseDaylight fields of Zone_Info (see the definitions in Section 2.1).

The zone record is used in User_Time. The date and time information in User_Time is broken down into fields for input and output. Both the time zone index and application of daylight savings time can be either explicitly specified or defaulted through use of the Zone_Info fields. The Weekday field is unused for input (see the definitions in Section 2.1).

2. Use

The definitions throughout this document are given in Pascal. If you are programming in the C language, please refer also to the document "C System Interfaces" in the *Accent Languages Manual*. If you are programming in the Lisp language, see the document "Lisp Interaction with the Accent Operating System" in the *Accent Lisp Manual*. When FORTRAN becomes available under Accent, the definitions will be the same as in the C language.

2.1. Type Definitions

These type and constant definitions are found in module TimeDefs in TimeDefs.Pas in LibPascal.

Internal_Time: a record containing the date and time in Greenwich Mean Time. This is the Smithsonian Institute's time standard. To optimize space usage, time is stored as an ordered pair. The first represents the number of weeks which have passed since 17-Nov-1858, when the Smithsonian time standard began. The second represents the number of milliseconds which have passed in the current week, which begins at 00:00 a.m. Wednesday.

```
type
    Internal_Time    = record
        Weeks        : integer;  { Number of weeks since
                                   17-Nov-1858 }
        MSecInWeek   : long;     { Number of milliseconds
                                   in that week }
        end;
```

Date_Fields: a record containing fields necessary for representing date information without respect to the time. Used in User_Time.

```
type
    Date_Fields = packed record
        Year    : integer;  { Such as 1982 }
        Month   : 1 .. 12;  { 1 = January, 12 = December }
        Day     : 1 .. 31;
        Weekday : 0 .. 6;   { 0 = Monday, 6 = Sunday
                              (output only) }
        end;
```

Time_Fields: a record containing fields necessary for representing time information without respect to the date. Used in User_Time.

```
type
    Time_Fields     = packed record
        Hour        : 0 .. 24;
        Minute      : 0 .. 59;
        Second      : 0 .. 59;
        Millisecond : 0 .. 999;
        end;
```

Zone_Info: this record allows the user to specify and receive time zone information. Both the zone number and application of daylight savings time may be either specified explicitly or defaulted, depending upon the settings of the UseTimeZone and UseDaylight bits. Used in User_Time.

```
type
    Zone_Info  = packed record
        TimeZone    : integer;  { Increasing minutes west from
                                  GMT.  GMT = 0,
                                  EST = 5*60,
                                  CST = 6*60, ...
                                  Used only if
                                  UseTimeZone is set. }
        UseTimeZone : boolean;  { True when TimeZone field is
                                  valid, else false when the
                                  system time zone is to be
                                  used. }
        Daylight    : boolean;  { True if daylight savings time
                                  is to be applied.  Used only
                                  if UseDaylight is set. }
        UseDaylight : boolean;  { True if Daylight savings field
                                  is valid, else false when the
                                  system default for daylight
                                  savings time application
                                  is to be used. }
        end;
```

User_Time: a record containing Date and Time information broken down into fields as the user would want to use it for input and output. Both the time zone index and application of daylight savings time can be either explicitly specified or defaulted through use of the Zone_Info fields. The Weekday field is unused for input.

```
type
    User_Time       = packed record
        Date        : Date_Fields;
        Time        : Time_Fields;
        Zone        : Zone_Info;
        end;
```

The following flag values may be ORed together to form TimeFormat values. They are used in the GetStringTime, T_IntToString, and T_UserToString procedures. Values are in octal.

```
const
    TF_Weekday      = #000001;  { If set, output the day of
                                  the week according to the
                                  setting of TF_FullWeekday,
                                  else don't output the day
                                  of the week }
    TF_FullWeekday  = #000002;  { If set, output full text for
                                  the weekday, else the
                                  3-letter abbreviation
                                  (Monday/Mon) }
    TF_NoDate       = #000004;  { If set, do not output date
                                  and ignore flags through
                                  TF_NoTime }
    TF_FullMonth    = #000010;  { If set, output full text for
                                  the month when the month
                                  is alphabetic, else the
                                  3-letter abbreviation
                                  (March/Mar) }
    TF_FullYear     = #000020;  { If set, output the year as a
                                  4-digit number, else the
                                  year is output as a 2-digit
                                  number if in the range
                                  1900-1999 (1982/82) }
```

The next six settings are mutually exclusive.

```
    TF_Dashes       = #000000;  { Output date as
                                  day-month-year
                                  (22-Mar-60) }
    TF_Spaces       = #000040;  { Output date as
                                  day month year
                                  (22 Mar 60) }
    TF_Reversed     = #000100;  { Output date as
                                  month day, year
                                 (Mar 22, 60) }
    TF_Slashes      = #000140;  { Output date as
                                  month/day/year
                                  (03/22/60) }

    { #000200 is reserved for future expansion}
    { #000240 is reserved for future expansion}

    TF_ANSI         = #000300;  { Output date according to
                                  ANSI X3.30-1971.  Also
                                  slightly affects time
                                  formatting.(600322) }
    TF_ANSI_Ordinal = #000340;  { Similar to TF_ANSI but
                                  3-digit day-of-year instead
                                  of month and day.
                                  (60082) }
```

```
TF_DateFormat    = #000340;  { A mask allowing examination
                               of the above. }

TF_NoTime        = #000400;  { If set, do not output time
                               and ignore flags through
                               TF_NoColumns }
```

The next two settings are mutually exclusive.

```
TF_NoSeconds     = #001000;  { If set, do not output the
                               seconds }

TF_Milliseconds = #002000;   { If set, output milliseconds
                               as hh:mm:ss.sss, else omit
                               them
                               (17:00:00.001/17:00:00) }

TF_12_Hour       = #004000;  { If set, output the time in
                               12-hour format with 'am'
                               or 'pm' following the time,
                               else output in 24-hour
                               format.  Note that exact
                               NOON outputs neither 'am'
                               nor 'pm' because 12:00am
                               is 0000 and 12:00pm is
                               2400.  Use of
                               TF_12_Hour with
                               TF_ANSI or
                               TF_ANSI_Ordinal is
                               supported but not
                               recommended for a number
                               of reasons.  If used with
                               either ANSI format,
                               however, the codes 'A',
                               'P', and 'N' are used
                               for am, pm, and noon,
                               respectively.
                               (5:00:00pm/17:00:00) }
TF_TimeZone      = #010000;  { If set, output the time
                               zone as -zzz after the
                               time, else omit it
                               (17:00:00-EDT/17:00:00) }

TF_NoColumns     = #040000;  { If set, output numeric
                               date/time quantities in
                               the smallest fields into
                               which they will fit.
                               If not set, the
                               date/time will be output
                               in fixed length fields,
                               thus making this format
                               appropriate for columnar
                               display.  Note that
                               TF_FullMonth and
                               TF_FullWeekday are
                               currently NOT padded with
```

```
                                          blanks, even if
                                          TF_NoColumns is off. }
    TF_BlankPad          = #020000;     { If set, pad fixed-width
                                          numbers with blanks
                                          instead of zeroes WHERE
                                          REASONABLE.  Ignored if
                                          TF_NoColumns is set. }

    TF_Never             = #100000;     { If set, allow the
                                          distinguished time
                                          value NEVER to be
                                          output as the string
                                          'Never', else signal an
                                          error }
```

The following flags are returned to indicate which fields of the date and time were present upon parsing a date/time string. The flags are ORed together to indicate that more than one item was found in the string. They are returned in the variable WhatIFound by T_StringToInt and T_StringToUser.

```
const
    TP_Weekday        = #000001; { Weekday present }
    TP_Date           = #000002; { Date present }
    TP_Time           = #000004; { Time present }
    TP_Zone           = #000010; { TimeZone present }
    TP_Never          = #000020; { Time input was NEVER }
    TP_RESERVED       = #177740; { Reserved for expansion }


  String_255:  The maximum length string.  Dates are parsed
               from such strings.

type
    String_255        = string[255];
```

2.2. Exception Definitions

The exception definitions are found in module Time, in the file TimeUser.Pas in LibPascal.

BadDateTime

This exception is raised if a bad date/time value is passed to any of the TimeServer routines, or if the TimeFormat flags (for conversion to String format) are invalid.

TimeNotInitialized

This exception is raised if the system date, time, and time zone have not been set. It will almost never be raised since these are set during system initialization.

2.3. Routine Definitions

The routine definitions are found in module Time, in the file TimeUser.Pas in LibPascal.

2.3.1. Setting the current time

```
procedure SetDateTime(
                ServPort : port;
                ITime    : Internal_Time
        );
```

Abstract:

Sets current time.

Parameters:

ServPort	TimePort (service port to Time Server, imported from PascalInit.Pas in LibPascal).
ITime	Internal_Time record for time to set.

Side Effects:

Sets current date and time.

Errors:

Raises TimeNotInitialized if system time zone and daylight switch have not been set.

2.3.2. Setting system defaults

```
procedure SetSystemZone(
                ServPort      : port;
                TimeZone      : integer;
                DSTWhenTimely : boolean
        );
```

Abstract:

Sets the system defaults for time zone and whether to use daylight savings time.

Parameters:

ServPort	TimePort (service port to Time Server imported from PascalInit.Pas in LibPascal).

TimeZone	System time zone to set (see the constant definitions in Section 2.1).
DSTWhenTimely	Whether to use daylight savings time during the USA daylight savings time interval.

2.3.3. Getting time in internal time format

```
function  GetDateTime(
                ServPort: Port
        ): Internal_Time;
```

Abstract:

Gets current time, in Internal_Time format.

Parameters:

ServPort	TimePort (service port to Time Server imported from PascalInit.Pas in LibPascal).

Returns:

Current time, in Internal_Time format.

Errors:

Raises TimeNotInitialized if system time has not been set.

2.3.4. Getting time in user time format

```
function  GetUserTime(
                ServPort: port
        ): User_Time;
```

Abstract:

Gets current time, in User_Time format.

Parameters:

ServPort	TimePort (service port to Time Server imported from PascalInit.Pas in LibPascal).

Returns:

Current time, in User_Time format, according to system time zone and daylight time defaults.

Errors:

Raises TimeNotInitialized if system time has not been set.

2.3.5. Getting time in string format

```
function  GetStringTime(
                ServPort    : port;
                TimeFormat  : integer
        ): String;
```

Abstract:

Gets current time, in string format.

Parameters:

ServPort TimePort (service port to Time Server imported from PascalInit.Pas in LibPascal).

TimeFormat
Format for the string (see the constant definitions in Section 2.1.

Returns:

Current time, according to system time zone and daylight time defaults, converted according to TimeFormat.

Errors:

Raises TimeNotInitialized if system time has not been set.

2.3.6. Converting internal to user time, with supplied zone

```
function  T_IntToZone(
                ServPort : port;
                ITime    : Internal_Time;
                WantZone : Zone_Info
        ): User_Time;
```

Abstract:

Converts internal time to user time, according to supplied time zone.

Parameters:

ServPort TimePort (service port to Time Server imported from PascalInit.Pas in LibPascal).

ITime Internal_Time record.

WantZone	Zone_Info record containing Zone and daylight desired.

Returns:

User_Time record representing ITime.

2.3.7. Converting internal to user time, with defaults

```
function T_IntToUser(
                ServPort  : Port;
                ITime     : Internal_Time
        ):User_Time;
```

Abstract:

Converts internal time to user time, according to system time zone and daylight time defaults.

Parameters:

ServPort	TimePort (service port to Time Server imported from PascalInit.Pas in LibPascal).
ITime	Internal_Time record.

Returns:

User_Time record representing ITime.

2.3.8. Converting user to internal time

```
function T_UserToInt(
                ServPort  : Port;
                UTime     : User_Time
):Internal_Time;
```

Abstract:

Converts user time to internal time.

Parameters:

ServPort	TimePort (service port to Time Server imported from PascalInit.Pas in LibPascal).
UTime	User_Time record.

Returns:

Internal_Time corresponding to UTime.

2.3.9. Converting user to string time

```
function  T_UserToString(
                ServPort   : port;
                UTime      : User_Time;
                TimeFormat : integer
          ): String;
```

Abstract:

Converts a user time record to a string representing the time, according to the conversion parameters.

Parameters:

ServPort	TimePort (service port to Time Server imported from PascalInit.Pas in LibPascal).
UTime	User_Time record.
TimeFormat	Format desired for the output (see the constant definitions in Section 2.1).

Returns:

A string representation of UTime.

2.3.10. Converting internal to string time

```
function  T_IntToString(
                ServPort   : Port;
                ITime      : Internal_Time;
                TimeFormat : Integer
         ): String;
```

Abstract:

Converts an internal time record to a string representing the time, according to the conversion parameters. The system defaults for time zone and daylight time are used.

Parameters:

ServPort	TimePort (service port to Time Server imported from PascalInit.Pas in LibPascal).
ITime	Internal_Time record.

TimeFormat
Format desired for the output (see the constant definitions in Section 2.1).

Returns:

A string representation of ITime.

2.3.11. Converting string to user time

```
function  T_StringToUser(
                ServPort    : port;
                STime       : String_255;
            var Index       : integer;
            var WhatIFound  : integer
          ): User_Time;
```

Abstract:

Converts a string to a user-time record.

Parameters:

ServPort	TimePort (service port to Time Server imported from PascalInit.Pas in LibPascal).
STime	String to be converted to time.
Index	Position in string to start scanning for time. Returns the first character past the end of the valid time string.
WhatIFound	Returns what was parsed from the time string (see the constant definitions in Section 2.1).

Returns:

User-Time record that STime represents.

Errors:

Raises BadDateTime if STime malformed

2.3.12. Converting string to internal time

```
function  T_StringToInt(
                ServPort   : port;
                STime      : String_255;
            var Index      : integer;
            var WhatIFound : integer
         ): Internal_Time;
```

Abstract:

Converts a string to an internal-time record.

Parameters:

ServPort	TimePort (service port to Time Server imported from PascalInit.Pas in LibPascal).
STime	String to be converted to time.
Index	Position in string to start scanning for time. Returns the first character past the end of the valid time string.
WhatIFound	Returns what was parsed from the time string (see the constant definitions in Section 2.1).

Returns:

Internal_Time record that STime represents.

Errors:

Raises BadDateTime if STime malformed

2.3.13. Returning "never"

```
function  T_Never(
                ServPort: Port
        ): Internal_Time;
```

Abstract:

Returns the internal time value representing "never".

Parameters:

ServPort	TimePort (service port to Time Server imported from PascalInit.Pas in LibPascal).

Returns:

Internal_Time record representing Never.

2.4. Valid Time Zones

Following is a list of valid time zones accepted by T_StringToUser and T_StringToInt.

```
GMT  Greenwich Mean Time
UT   Universal Time (same as Greenwich
         Mean Time)

NST Newfoundland Standard Time  (-3:30 hrs)
AST Atlantic Standard Time      (-4 hrs)
ADT Atlantic Daylight Time
EST Eastern Standard Time       (-5 hrs)
EDT Eastern Daylight Time
CST Central Standard Time       (-6 hrs)
CDT Central Daylight Time
MST Mountain Standard Time      (-7 hrs)
MDT Mountain Daylight Time
PST Pacific Standard Time       (-8 hrs)
PDT Pacific Daylight Time
YST Yukon Standard Time         (-9 hrs)
YDT Yukon Daylight Time
HST Hawaii-Alaska Standard Time (-10 hrs)
HDT Hawaii-Alaska Daylight Time
BST Bering Standard Time        (-11 hrs)
BDT Bering Daylight Time
```

2.5. String Times

String times are in the (very general) format:

```
Weekday                (* Weekday - any unique abbreviation *)

mo/dd/yyyy       |     (* Date - digits for day, month, year *)
Month dd, yyyy   |     (* any unique abbreviation of Month *)
dd Month yyyy    |
dd-Month-yyyy

hh:mm[:ss[.ttt]]       (* Time *)
[am | a | pm | p ]

TimeZone |             (* Timezone - one of the time zones listed
                          in Section 2.4 *)
+|- hours[:minutes]    (* hours east (+) or west (-) of GMT *)
```

String times can be supplied or omitted in any combination, as long as they are specified in the order: Weekday, Date, Time, Timezone.

Any time output by T_IntToString, T_UserToString, or GetStringTime is acceptable as a string time in T_StringToUser or T_StringToInt.