Date Manipulation Routines

by Thomas Wm. Madron
Email: info@ewc-inc.com

Table of Contents

LICENSE AGREEMENT

This license agreement covers your use of the Enterprise-Wide Computing, Inc. EWCDATE.DLL, its source code, documentation, and executable files, hereinafter referred to as "the Product".

The Product is Copyright (c) 1998-99 by Enterprise-Wide Computing, Inc. You may use it and distribute it according to this following License Agreement. If you do not agree with these terms, please remove the Product from your system. By incorporating the Product in your work or distributing the Product to others you implicitly agree to these license terms.

This License Agreement covers the current version of The Product. Enterprise-Wide Computing, Inc. reserves the right to modify the terms of this License Agreement at any moment, and without prior notification, in future releases of The Product.

STATEMENT OF COPYRIGHT

The Product is, and remains, copyright 1998-98 by Enterprise-Wide Computing, Inc.

GENERAL CONDITIONS

The Product consists of a ready-to-use DLL (EWCDATE.DLL), the source code in BASIC that can be compiled by PowerBasic PBDLL5 and later, an include file containing all appropriate DECLARE statements for use with the PowerBasic 32-bit compilers, PBDLL5 and later, and PB/CC 1 and later, and the documentation for the package (EWCDATE.PBF).

You may freely use and distribute the Product DLL with any program that uses components of the DLL. The source code is provided to assist the user in modifying or expanding the components of the DLL. The source code, as distributed, remains copyrighted by Enterprise-Wide Computing, Inc. except as other copyrights may apply.

RIGHTS OF USAGE

You may freely and at no additional cost use the Product in any project, commercial, academic, military, or private, so long as you respect the License Agreement. The License Agreement does not affect any software except the Product. In particular, any application that uses the Product does not itself fall under the License Agreement. EWCDATE.DLL may be freely distributed with any product or project using components of EWCDATE.DLL.

You may modify any part of the Product, including sources and documentation, except this License Agreement, which you may not modify.

You must clearly indicate any modifications at the start of each source file. The user of any modified Product code must know that the source file is not original.

At your discretion, you may rewrite or reuse any part of the Product so that your derived code is not obviously part of the Product. This derived code does not fall under the Product License Agreement directly, but you must include a credit at the start of each source file indicating the original authorship and source of the code, and a statement of copyright as follows:

"Parts copyright (c) 1998-99 by Enterprise-Wide Computing, Inc."

RIGHTS OF DISTRIBUTION

You may not redistribute the original Product in any form, either free or for a fee, without purchasing additional licenses except EWCDATE.DLL in association with any program or product using the components of the DLL.

At no time will Enterprise-Wide Computing, Inc. associate itself with any distribution of the Product except that supplied from Enterprise-Wide Computing, Inc.

DISCLAIMER OF WARRANTY

The Product is provided as a commercial product in the hope that it will be useful. It is provided "as-is", without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to determining the suitability, quality and performance of the Product is with you. Should the Product prove defective, the full cost of repair, servicing, or correction lies with you.

TECHNICAL SUPPORT

Limited technical support can be had from support@ewc-net.com.

Published by Enterprise-Wide Computing, Inc.
URL: http://ewcnet.com; Email: info@ewc-inc.com
June 27, 1999

Package Contents

This package of date routines consists of the following files:

EWCDATE.DLL	32-bit Windows DLL containing all routines described in this document.
EWCDATE.BAS	Source code for EWCDATE.DLL.
EWCDATE.INC	Include file for PowerBasic Console Compiler (pbcc).
EWCDATE.PDF	Documentation in PDF file format. Requires AcroRead 2.1 or later.
TSTDT.BAS	Test program exercising all functions and procedures in this package.
TEST.BAS	Validity testing program for routines that calculate a Julian Number or its functional equivalent.
README.TXT	Installation instructions.

Installation

After unzipping the distribution archive into a temporary directory, copy/move EWCDATE.DLL to C:\windows\system (for Win95/98) or C:\winnt\system32 (for Win NT), or wherever you have put your Windows system directory. Copy/move the other files to any convenient location. This package does not make use of any other libraries, although you might note that there are additional date functions available through the Windows API.

To use the subroutines in this package with the PowerBasic PBCC (Console Compiler) or PBDLL simply add the following include statement at the top of your program(s):

```
$INCLUDE EWCDATE.INC
```

The EWCDATE.INC file must be where it can be found by the compiler or accessed with a fully qualified file specification on the include statement. EWCDATE.INC contains correct declare statements for all routines in EWCDATE.DLL.

Introduction

This file documents a 32-bit windows dll that contains a total of 37 date manipulation and support routines. The techniques and algorithms used in the implementation of these routines can be found in the bibliography at the end of this file. We have been collecting these routines since the early 1980s and all have passed through a number of versions of Basic, now implemented for compilation by PowerBasic's DLL compiler, v. 6. Some of the routines even started life in other languages, usually Fortran, but sometimes in Pascal.

There is some overlapping, in terms of functionality, of some of the routines. This has arisen as a byproduct of the sources of various of the routines, plus the fact that there are several ways to deal with the same or similar issues.

Issues in Date Manipulations

The most common problem in manipulating dates is to find the length of time, measured in days, between two dates. Various disciplines need to know the elapsed time between historical events, measured in some standard fashion. This is a common problem in astronomy, information technology, and sometimes in history. The general solution to this issue is to calculate the dates in question as a sequential number of days from some starting point, do whatever addition or subtraction is necessary, then convert the resulting sequential number back to a common format of month, day, and year. These sequential numbers are often, but incorrectly, termed Julian Calendar dates. Only one specific method for calculating dates in terms of a sequence of days is properly called a Julian period or date.

The Julian period, a chronological system now used chiefly by astronomers and based on the consecutive numbering of days from Jan. 1, 4713 BC. Not to be confused with the Julian calendar, the Julian period was proposed by the scholar Joseph Justus Scaliger in 1583 and named by him for his father, Julius Cesar Scaliger. Joseph Scaliger proposed a period of 7,980 years of numbered days to be used in determining time elapsed between various historical events otherwise recorded only in different chronologies, eras, or calendars. The length of 7,980 years was chosen as the product of 28 times 19 times 15; these respectively, are the numbers of years in the so-called solar cycle of the Julian calendar in which dates recur on the same days of the week; the lunar or Metonic cycle, after which the phases of the Moon recur on a particular day in the solar year, or year of the seasons; and the cycle of indiction, originally a schedule of periodic taxes or government requisitions in ancient Rome. The epoch, or starting point, of 4713 BC was chosen as the nearest past year in which the three cycles began together.

The Julian period or date is often confused with the Julian Calendar, also called OLD STYLE CALENDAR, a dating system established by Julius Caesar as a reform of the Roman republican calendar. Caesar, advised by the Alexandrian astronomer Sosigees, made the new calendar solar, not lunar, and he took the length of the solar year as 365 1/4 days. The Gregorian calendar, also called NEW STYLE CALENDAR, is the solar dating system now in general use. It was proclaimed in 1582 by Pope Gregory XIII as a reform of the Julian calendar. By the Julian reckoning, the solar year comprised 365 1/4 days; the intercalation of a "leap day" every four years was intended to maintain correspondence between the calendar and the seasons. A slight inaccuracy in the measurement (the solar year comprising more precisely 365 days, 5 hours, 48 minutes, 46 seconds) caused the calendar dates of the seasons to regress almost one day per century.

Joseph Scaliger proposed his system in order to determine the time elapsed between various historical events otherwise recorded only in different chronologies, eras, or calendars. This is precisely the reason why the system is used by astronomers, and today, in information technology, for significant date manipulation. Technically, a Julian period or date is one that stipulates the number of days from January 1, 4713 before the common era (bce) to the date of interest. Such a Julian date becomes confused with the Gregorian calendar because we currently use that calendar and are typically interested in getting back to real dates. Confusion arises, however, when we calculate a Gregorian date for any period before 1582 and then try to synchronize such a date with dates recorded in documents prior to 1582. The reason, of course, is that until 1582 people in the Western world, at least, used the old Julian calendar. The material concerning julian numbers is taken from "Julian Period." *Britannica CD*. Version 97. Encyclopaedia Britannica, Inc., 1997.

Overview of the Date Routines

There are several collections of date routines included. Two of those collections have close internal dependencies. One is derived from Lewis Rosenfelder, *Basic Faster and Better* (Upland, CA: IJG Inc., 1981), pp. 109-10 and the other from Michael A. Covington, "A Calendar for the Ages," *Pc Tech Journal*, vol. 3, no. 12, Dec. 85, pp. 136ff. In general, these groups of routines have dependencies from one routine to another. If you use any of these routines be aware that they might not be easily used with other, more generalized functions or sub programs. A third group was presented in John P. Grillo and J. D. Robertson in two books, *Subroutine Sandwich* (New York: John Wiley & Sons, 1983, p. 32, and *More Subroutine Sandwich* (New York: John Wiley & Sons, 1983, p. 36. Grillo and Robertson site J. D. Robertson, "Remark on Algorithm 398", *Communications of the ACM*, Vol. 15, No. 10, 1972, p. 918. The fourth group of routines were written by Thomas Wm. Madron. Neither the Grillo/Robertson nor the Madron collections assume any dependencies on other routines.

Fundamental to date calculations is the ability to take one or more dates, convert the date(s) into a sequential number based on some (often arbitrary) starting point, then be able to convert those sequential numbers back to a date. Virtually all other date computations are based on the sequential number, often referred to as a Julian Number. As was noted in the introduction, not all sequential data numbers are Julian Numbers, but all serve essentially the same purpose. The Covington, Grillo, and Rosenfelder collections all have this functional capability. The validity of the computational technique can be easily tested by simply calculating a Julian Number from an arbitrary date, then reproducing the input date from the Julian Number. In Table 1 are the results for such a test, using ewcdate.dll, for a series of dates from 1699 to 2099, with random months and days:

Table 1: Julian and Computational Date Validation

Input	Source	Julian#	Result
5/31/1699	Grillo	2341758	05/31/1699
12/22/1799	Grillo	2378487	12/22/1799
10/30/1899	Grillo	2414958	10/30/1899
11/29/1999	Grillo	2451512	11/29/1999
8/30/2099	Grillo	2487946	08/30/2099
4/2/1699	Covington	-102540	4/2/1699
8/4/1799	Covington	-65892	8/4/1799
11/3/1899	Covington	-29277	11/3/1899
10/26/1999	Covington	7239	10/26/1999
11/10/2099	Covington	43779	11/10/2099
2/12/1699	Rosenfelder	620602	2/12/1699
10/26/1799	Rosenfelder	657383	10/26/1799
6/4/1899	Rosenfelder	693764	6/4/1899
11/27/1999	Rosenfelder	730465	11/27/1999
4/4/2099	Rosenfelder	766753	4/4/2099

The first column is the input date, the second is the collection used for the computation, the third column is the Julian, or Julian-like number calculated from the input date, and the fourth column is the date derived from the Julian (or Julian-like) number. The input and output formats are in the form used in the United States: month/day/year. The Grillo function, *Julian*, comes closest to the original Julian Number procedure with a starting date of approximately Jan. 1, 4713 BC. While all these collections appear accurate, the Rosenfelder collection should be used with caution because it does not use a completely generalized approach for dealing with leap years.

The Rosenfelder Collection

The following functions were based on Lewis Rosenfelder:
'

FUNCTION CompDate	Computational Date (Julian-like)
FUNCTION CompDayNumber	Day of year
FUNCTION CompDayOfMonth	Day of month
FUNCTION CompMonth	Month
FUNCTION CompYear	Year
FUNCTION CompDayName	Day Name
FUNCTION JulianDate	Sequential day within a given year. Not a true Julian number.

Those routines starting with "Comp" require that a computational date be calculated with FUNCTION CompDate.

Largely because Rosefelder did not bother to provide a generalized approach to dealing with leap years the routines may not be reliable outside the range of 1901-2099, although, as may be seen in Table 1, they were correct outside that range for the sample dates. They compensate for leap years within this period but not in a wholly generalized manner. The Rosenfelder routines are highly dependent on what he called a computational date, calculated with the function CompDate. CompDate produces a Julian-like number that can be used in date calculations providing that the dates fall between 1901 and 2099 ce (in the common era). The problem with the Rosenfelder calculations can arise with years that are divisible by 4, but are not leap years.

Years divisible by 4 are leap years except for years divisible by 100 unless those years are also divisible by 400. Thus, 2000 is a leap year, but 1900 and 2100 are not. The reason for this adjustment is that from at least 730 AD it was known that the solar year was somewhat short of 365.25 days, the assumed length of the year under the Julian calendar. When Pope Gregory XIII instituted calendar reform in 1582, as part of that reform, he adopted the formula noted to keep the calendar closer to the actual solar year. Because the solar year is shortening, astronomers today keep the Gregorian calendar in line by making a one second adjustment, as needed, normally on December 31 at midnight, whenever the accumulation of errors nears one second. This function takes all of this into consideration.

The Covington Collection

The purpose of the following routines is to calculate long-range dates for the more distant past and future. The original routines were written in Pascal, then translated into Fortran, then into Turbo BASIC from the Fortran and finally reworked into PowerBasic.

FUNCTION DayNum	Day number from 1 Jan 1980
SUB CalDat	Date from (derived from DayNum)
FUNCTION WeekDay	Day of week (derived from DayNum)
FUNCTION rgtALIGN$	Private utility
FUNCTION Dfrac	Private utility
FUNCTION Dint	Private utility
FUNCTION Floor	Private utility

For this collection the computational date is provided by FUNCTION DayNum using 1 Jan 1980 as the base for the system. Dates before 1 Jan 1980 will generate a negative DayNum, those after a positive DayNum. SUB CalDat recalls the normal date from the number calculated by DayNum.

The Grillo and Robertson Collection

In their books *Subroutine Sandwich* and *More Subroutine Sandwich*, Grillo and Robertson presented several routines for date manipulation. Unlike the previous collections, all of these are independent of one another. Through translation those included in EWCDATE.DLL deriving from Grillo and Robertson are:

FUNCTION julian	Julian Number
SUB revjulian	Returns month, day, and year from julian
SUB JulToDate	Returns date string from julian
FUNCTION zeller	Day of week (0 - 6) using Zeller s Congruence
FUNCTION DayOfWeek	Day of week (1 - 7) using FUNCTION zeller
FUNCTION LeapYear	Calculates leap year (0=no; 1=yes)
FUNCTION DayOfYear	Sequential day within a given year.

Although Grillo and Robertson provide a subroutine that returns a normal date from the Julian Number calculated by FUNCTION julian, a separate procedure (JulToDate) by an unknown author is also used in this DLL to return a date string (mm/dd/[-]yyyy).

The Madron Collection

In addition to the basic computational routines from the collections already described, a number of additional utility functions and procedures have been written over the years by Thomas Wm. Madron. Most, though not all, use the computational subprograms in the other collections to do the actual date manipulations:

FUNCTION LongDayName	Returns the full name of the day in week.
FUNCTION ShortDayName	Returns a 3-character day in week abbreviation.
FUNCTION LongMonthName	Returns the full name of the month.
FUNCTION ShortMonthName	Returns a 3-character month abbreviation.
SUB CurrentDate	Returns current Month&, Day&, Year&.
FUNCTION Today	Returns current date in standard English.
SUB ReportDate	Returns any date in standard English.
FUNCTION FirstSunday	Returns day of first Sunday for month and year.
FUNCTION DaysInMonth	Given Month& and Year&, returns number of days in Month&.
SUB GetTodaysDate	Returns elements of date as strings for display.
FUNCTION AnyDay	Returns Day& within Month& and Year& of the Nth day of week (i.e., 3rd Tuesday, 4th Wednesday).
FUNCTION DaySavStrt	Returns the day of year of the starting of daylight savings time (U.S.).
FUNCTION DaySavEnd	Returns the day of year of the ending of daylight savings time (U.S.).
FUNCTION DayLightSavings	Determines whether a given date is in standard or daylight savings time (0=standard; -1=daylight savings time).
FUNCTION StandardDate	Returns a string containing a date in RFC 822 standard format.
FUNCTION MonthInYear	Returns the month number for a given year and day number within the year.
FUNCTION MonthFromName	Returns the month number from its English name.
SUB ParseEnglshDate	Returns month, day, and year numbers from a date string expressed in either U.S. or European formats in English.

A Note of Caution

When using these routines for date computations, do not mix the computational date functions of one collection with the retrieval functions of another collection. You cannot, for example, calculate a

computational date with FUNCTION CompDate (Rosefelder) and retrieve the Month, Day, and Year from that number with FUNCTION revjulian (Grillo and Robertson). The three computational date functions in this collection are:

FUNCTION CompDate	Rosenfelder
FUNCTION DayNum	Covington
FUNCTION julian	Grillo and Robertson

Many people in Information Technology often confuse a frequently used measurement of time, the number of days from January 1 of the current year to some date of interest (sequential day in year) with Julian numbers. While such a measure can be used for some of the same purposes that Julian numbers are used, these are clearly not Julian numbers.

Using EWCDATE.DLL

Including *EWCDATE.inc*

To use these date functions and procedures copy EWCDATE.DLL into c:\windows\system or c:\winnt\system32 (or wherever your windows directory resides). You should be able access these routines from any language that can make use of standard 32-bit DLLs. For the PowerBasic Console Compiler, include EWCDATE.INC at the top of your program:

```
$INCLUDE  EWCDATE.inc
```

The include file contains all the declare statements necessary to use any routine in EWCDATE.DLL. Since there may dependencies from one function or procedure to another, use the entire include file without trying to edit it for only those functions and/or procedures you are using.

Calling the Functions and Procedures

When numeric date information is required by a routine, or returned by a routine, the values are normally passed as long integers (i.e., Month&, Day&, Year&). The three values most frequently encountered are:

Month& = long integer containing a month number (1-12, 1=January).
Day& = long integer containing a day-of-month number (1-31).
Year& = long integer containing a year (yyyy).

Julian and Julian-like Numbers

Date calculations must, in some way or another, start out with the calculation of a julian number or something like a julian number. In this packages there are several such functions:

FUNCTION julian	Grillo	Calculate a true julian date.
FUNCTION DayOfYear	Grillo	Calculate the sequential day of the year.
FUNCTION DayNum	Covington	Number of days elapsed since 1980 jan 0 (1979 dec 31).
FUNCTION CompDate	Rosenfelder	Calculate a computational date for later use in related subprograms.
FUNCTION JulianDate	Rosenfelder	Calculate a sequential day number within a year.
FUNCTION CompDayNumber	Rosenfelder	Recall day number within year from computational date.

The primary similarity among all these functions is the ability to use the number calculated as the basis for various date calculations. For example, if we wanted to find how many days will elapse between two dates within a given year, we could use DayOfYear, JulianDate, or CompDayNumber to calculate the sequential day number of each date, them simply subtract the two numbers. In information processing the sequential day number within a year is often, but incorrectly called a julian date. In fact, Rosenfelder s JulianDate number is so misnamed.

In this group of functions julian, DayNum, and CompDate all provide more generalized sequential numbers allowing date computations across years.

julian

Function

Purpose: To calculate a true julian date.

Author/Reference: John P. Grillo and J. D. Robertson, *More Subroutine Sandwich* (New York: John Wiley & Sons, 1983, p. 36. Grillo and Robertson site J. D. Robertson, "Remark on Algorithm 398", *Communications of The ACM*, Vol. 15, No. 10, 1972, p. 918. (TWM, 4/3/88).

Syntax: j# = julian (Month&, Day&, Year&)

```
DECLARE FUNCTION julian ALIAS "julian" (Month&, Day&, Year&) export AS
        DOUBLE
```

Parameters: Parameters are returned by CurrentDate.
Month& = long integer containing a month number (1-12, 1=January).
Day& = long integer containing a day-of-month number (1-31).
Year& = long integer continaing a year (yyyy).

Remarks:

Example:

DayNum

Function

Purpose: Number of days elapsed since 1980 jan 0 (1979 dec 31) given year as full four digit number.

Author/Reference: Michael A. Covington, "A Calendar for the Ages," *PC Tech Journal*, vol. 3, no. 12, Dec. 85, pp. 136ff.

Syntax:

```
dn& = DayNum (Month&, Day&, Year&)

DECLARE FUNCTION DayNum ALIAS "DayNum" (Month&, Day&, Year&)
        export AS LONG
```

Parameters:
Month& = long integer containing a month number (1-12, 1=January).
Day& = long integer containing a day-of-month number (1-31).
Year& = long integer continaing a year (yyyy).

Remarks:
Requires the following functions local to date.dll:
FUNCTION Floor
FUNCTION Dint
The use of these functions is transparent to the programmer.

Dates prior to January 1, 1980, will be returned as negative numbers.

Example:

DayOfYear

Function

Purpose: To calculate the sequential day of the year taking into account Leap Years.

Author/Reference: John P. Grillo, and J. D. Robertson, *Subroutine Sandwich* (New York: John Wiley & Sons, 1983, p. 36.

Syntax:

```
d& = DayOfYear (Month&, Day& Year&)

DECLARE FUNCTION DayOfYear ALIAS "DayOfYear" (Month&, Day&,
        Year&) export AS LONG
```

Parameters:
Month& = long integer containing a month number (1-12, 1=January).
Day& = long integer containing a day-of-month number (1-31).
Year& = long integer continaing a year (yyyy).

Remarks: Requires FUNCTION LeapYear (Year&)

Example:

CompDate

Function

Purpose: To calculate a computational date for later use in related subprograms based on Rosenfelder.

Author/Reference: Based on functions taken from Lewis Rosenfelder, *Basic Faster and Better* (Upland, CA: IJG Inc., 1981), pp. 109-10.

Syntax:

```
x& = CompDate(month&, day&, year&)

DECLARE FUNCTION CompDate ALIAS "CompDate" (month&, day&, year&)
        export as long
```

Parameters:
Month& = long integer containing a month number (1-12, 1=January).
Day& = long integer containing a day-of-month number (1-31).
Year& = long integer continaing a year (yyyy).

Remarks: This function produces a long integer number functionally (but not computationally) similar to a julian number. Valid for dates from 1901 through 2099. Takes leap years into account.

Example:

JulianDate

Function

Purpose: To calculate a sequential day number within a year (hence, not a true julian number).

Author/Reference: Based on functions taken from Lewis Rosenfelder, *Basic Faster and Better* (Upland, CA: IJG Inc., 1981), pp. 109-10.

Syntax:

```
x& = JulianDate (month&, day&, year&)

DECLARE FUNCTION JulianDate ALIAS "JulianDate" (month&, day&, year&)
        export as long
```

Parameters:
Month& = long integer containing a month number (1-12, 1=January).
Day& = long integer containing a day-of-month number (1-31).
Year& = long integer continaing a year (yyyy).

Remarks: Notwithstanding the name, this function calculates only the day number within a given year (from 1901 through 2099), not a true julian number.

Example:

CompDayNumber

Function

Purpose:	Recall day number within year from computational date.
Author/Reference:	Based on functions taken from Lewis Rosenfelder, *Basic Faster and Better* (Upland, CA: IJG Inc., 1981), pp. 109-10.
Syntax:	x& = CompDayNumber(d&)
	DECLARE FUNCTION CompDayNumber ALIAS "CompDayNumber"(d&) export as long
Parameters:	d& = CompDate(month&, day&, year&)
Remarks:	Requires the use of **FUNCTION CompDate**. Is equivalent to **FUNCTION JulianDate** but retrieves the day number from the computational date.
Example:	

Retrieving a Date from a Julian Number

One of the things we might wish to do, given a Julian number, is to retrieve a readable, common date from the julian number. There are three routines for accomplishing this, depending on what "julian" routine was used to obtain the julian number:

FUNCTION CompYear	Retrieves a year from a computational date.
FUNCTION CompMonth	Retrieves a month number from year and day number.
FUNCTION CompDayOfMonth	Retrieves a monthly day number from month, year, and day number.
SUB JulToDate	Retrieves date from a Julian number calculated with FUNCTION julian.
SUB CalDat	Retrieves date from number calculated with FUNCTION DayNum

Usage notes for CompYear, JulToDate and CalDat follow:

CompYear

Function

Purpose:	To retrieve a year from a computational date (see CompDate, above).
Author/Reference:	Based on functions taken from Lewis Rosenfelder, *Basic Faster and Better* (Upland, CA: IJG Inc., 1981), pp. 109-10.
Syntax:	x& = CompYear(d&)
	DECLARE FUNCTION CompYear ALIAS "CompYear" (d&) export as long
Parameters:	d& = CompDate(month&, day&, year&)
Remarks:	Requires the use of **FUNCTION CompDate**. Retrieves the year from the computational date.
Example:	

CompMonth

Function

Purpose:	To retrieve a month number from a computational day number (see CompDayNumber, above) and year (see CompYear, above).
Author/Reference:	Based on functions taken from Lewis Rosenfelder, *Basic Faster and Better* (Upland, CA: IJG Inc., 1981), pp. 109-10.
Syntax:	x& = CompMonth(cmpdaynum&, year&) DECLARE FUNCTION CompMonth ALIAS "CompMonth" (cmpdaynum&, year&) export as long
Parameters:	m& = CompMonth (cmpdaynum&, year&)
Remarks:	Requires the use of **FUNCTION CompDayNumber** and possibly **CompYear**.
Example:	

CompDayOfMonth

Function

Purpose:	Recall day of month from year, month, and day within year.
Author/Reference:	Lewis Rosenfelder, *BASIC FASTER AND BETTER* (Upland, CA: IJG Inc., 1981), pp. 109-10.
Syntax:	d& = CompDayOfMonth (year&, month&, cmpdaynum&) Cmpdaynum& is calculated using CompDate, another of Rosenfelder routines. DECLARE FUNCTION CompDayOfMonth ALIAS "CompDayOfMonth" (year&, month&, cmpdaynum&) export as long
Parameters:	year& - four digit long integer containing the year of interest. month& - long integer containing the month number of interest. cmpdaynum& - long integer containing Rosenfelder s computational date.
Requires:	FUNCTION CompDate
Remarks:	
Example:	

JulToDate

Procedure

Purpose:	To convert the Julian number of a date into it's date form ("mm-dd-[-]yyyy")
Author/Reference:	Unknown.
Syntax:	call JulToDate (JulianNumber#, ResultDate$) DECLARE SUB JulToDate ALIAS "JulToDate" (JulianNumber#, ResultDate$) EXPORT
Parameters:	JulianNumber# = Double precision real number containing a Julian number. Calculated with FUNCTION julian. ResultDate$ = String containing the date referenced by JulianNumber#.
Remarks:	Requires the following functions: FUNCTION Julian FUNCTION rgtAlign$ If the year is negative, it is bce (before the common era).
Example:	

CalDat

Procedure

Purpose: Inverse of daynum: given date finds year, month, day.

Author/Reference: SOURCE: Michael A. Covington, "A Calendar for the Ages," *PC Tech Journal*, vol. 3, no. 12, Dec. 85, pp. 136ff.

Syntax:

```
call CalDat (DayNum1&, Month&, Day&, Year&)

DECLARE SUB CalDat ALIAS "CalDat" (DayNum1&, Month&, Day&, Year&)
        EXPORT
```

Parameters:

DayNum1& = long integer containing the sequential day number
calculated with FUNCTION DayNum.
Month& = Returned as long integer containing a month number (1-12, 1=January).
Day& = Returned as long integer containing a day-of-month number (1-31).
Year& = Returned as long integer continaing a year (yyyy).

Remarks: Requires FUNCTION Dint.

Example:

Other Date Counting Utility Functions

When manipulating dates it is often necessary to know the sequential day number of a particular date, the number of days in a given month/year, and whether or not a given year is a leap year. Four functions provide this information:

FUNCTION zeller
FUNCTION DayOfWeek
function DaysInMonth
FUNCTION LeapYear

Zeller implements a well-known algorithm called zeller s congruence. The result of the application of this algorithm to a given date is a day number in the range of 0 - 6 where 0 = Sunday. **DayOfWeek** provides the same information, also using zeller s congruence, but provides the day number in the range 1 - 7 where 1 = Sunday. The number of days in a given month/year, is found by **DaysInMonth**, which compensates for leap years. **LeapYear** determines whether a given year is a leap year.

Detailed application notes follow:

zeller

Function

Purpose: To determine the day of the week for any date. Sometimes called Zeller's Congruence.

Author/Reference: John P. Grillo, and J. D. Robertson, *Subroutine Sandwich* (New York: John Wiley & Sons, 1983, p. 32.

Syntax: x& = zeller (month&, day&, year&) , where
0 = Sunday . . . 6 = Saturday

DECLARE function zeller ALIAS "zeller" (month&, day&, year&) export as long

Remarks:

Example:

DayOfWeek

Function

Purpose:	To determine the day of the week for any date.
Author/Reference:	Modified version of **zeller** (see *zeller*).
Syntax:	dw& = DayOfWeek (Month&, Day& Year&) 1 = Sunday . . . 7 = Saturday DECLARE FUNCTION DayOfWeek ALIAS "DayOfWeek" (month&, Day&, Year&) export AS LONG
Parameters:	Month& = long integer containing a month number (1-12, 1=January). Day& = long integer containing a day-of-month number (1-31). Year& = long integer continaing a year (yyyy).
Remarks:	see zeller (Month&, Day&, Year&) for further documentation. These are identical in function although DayOfWeek returns days as 1-7 while zeller returns days as 0-6. Both start counting with Sunday.
Example:	

DaysInMonth

Function

Purpose:	To return the number of days in month& for year&. This function accurately compensates for leap years.
Author/Reference:	Thomas Wm. Madron (1998)
Syntax:	nd& = DaysInMonth (month&, year&)
	DECLARE function DaysInMonth ALIAS "DaysInMonth" (month&, year&) export as long
Parameters:	Month& = long integer containing a month number (1-12, 1=January). Year& = long integer continaing a year (yyyy).
Requires:	FUNCTION LeapYear
Remarks:	
Example:	

LeapYear

Function

Purpose:	To determine if Year& is a Leap Year
Author/Reference:	John P. Grillo, and J. D. Robertson, *Subroutine Sandwich* (New York: John Wiley & Sons, 1983, p. 40.
Syntax:	l& = LeapYear (year&) DECLARE FUNCTION LEAPYEAR ALIAS "LeapYear" (year&) EXPORT AS LONG
Parameters:	Year& = long integer continaing a year (yyyy).
Remarks:	Years divisible by 4 are leap years except for years divisible by 100 unless those years are also divisible by 400. Thus, 2000 is a leap year, but 1900 and 2100 are not. The reason for this adjustment is that from at least 730 AD it was known that the solar year was somewhat short of 365.25 days, the assumed length of the year under the Julian calendar. When Pope Gregory XIII instituted calendar reform in 1582, as part of that reform, he adopted the formula noted to keep the calendar closer to the actual solar year. Because the solar year is shortening, astronomers today keep the Gregorian calendar in line by making a one second adjustment, as needed, normally on December 31 at midnight, whenever the accumulation of errors nears one second. This function takes all of this into consideration.
Example:	

Labeling Days and Months

Given dates of some kind, it is frequently necessary provide labels (specifically, day names and month names) for display. In this DLL there are five functions that provide labeling capabilities:

FUNCTION LongDayName
FUNCTION ShortDayName
FUNCTION LongMonthName
FUNCTION ShortMonthName
FUNCTION CompDayName

The first four are quite generalized. The long and short day name functions provide either a full day name (i.e., "Monday") or an abbreviated 3-character day name (i.e., "Mon"). The only parameter required for these functions is a day number where 1=Sunday and 7=Saturday. Similarly, the long and short month name functions provde full or abbreviated month names. The single parameter required is a month number where 1=January.

The CompDayName function is one of Rosenfelder's and requires not a day name but the computational date calculated with FUNCTION CompDate. It is, therefore, less generalized than the other labeling functions.

Usage notes for the labeling functions follow:

LongDayName

Function

Purpose: To return a full day name for day-of-week.

Author/Reference: Thomas Wm. Madron (1998)

Syntax: d$ = LongDayName (day&)

DECLARE FUNCTION LongDayName ALIAS "LongDayName" (day&)
EXPORT AS STRING

Parameters: day& = long integer containing a day-of-week number (1-7, 1=Sunday).

Remarks:

Example:

ShortDayName

Function

Purpose:	To return a short day-of-week name.
Author/Reference:	Thomas Wm. Madron (1998)
Syntax:	d$ = ShortDayName (day&)
	DECLARE FUNCTION ShortDayName ALIAS "ShortDayName" (day&) export as string
Parameters:	day& = long integer containing a day-of-week number (1-7, 1=Sunday).
Remarks:	
Example:	

LongMonthName

Function

Purpose:	To return a full month name.
Author/Reference:	Thomas Wm. Madron (1998)
Syntax:	m$ = LongMonthName (month&)
	DECLARE FUNCTION LongMonthName ALIAS "LongMonthName" (month&) export as string
Parameters:	month& = long integer containing a month number (1-12, 1=January).
Remarks:	
Example:	

ShortMonthName

Function

Purpose:	To return a short month name.
Author/Reference:	Thomas Wm. Madron (1998)
Syntax:	m$ = ShortMonthName (month&)
	DECLARE FUNCTION ShortMonthName ALIAS "ShortMonthName" (month&) export as string
Parameters:	month& = long integer containing a month number (1-12, 1=January).
Remarks:	
Example:	

Specialized Utility Routines

It is sometimes necessary to find out very specific things about dates or to display them in some particular fashion. There are several routines that do just that:

SUB CurrentDate
SUB GetTodaysDate
FUNCTION Today
FUNCTION StandardDate
FUNCTION FirstSunday
FUNCTION AnyDay

CurrentDate converts the current date (acquired from the computer system) to month, day, and year expressed as long integers. When dates are actually displayed, it is often useful to have the relevant numbers converted to names or other strings. This is accomplished with **GetTodaysDate** for the current date. **Today** reports the current date in standard Englsih. In a similar fashion, **StandardDate** provides the current date in RFC 822 format. **FirstSunday** and **AnyDay** provide the dates for the first Sunday in a given month/year and the date for any nth day (third Thursday, for example) in a month.

Detail application notes follow:

CurrentDate

Procedure

Purpose:	To return the current date as m&, d&, and y&.
Author/Reference:	Thomas Wm. Madron (1998)
Syntax:	call CurrentDate (m&, d&, y&) or CurrentDate m&, d&, y& DECLARE SUB CurrentDate ALIAS "CurrentDate" (m&, d&,y&) EXPORT
Parameters:	Parameters are returned by CurrentDate. m& = long integer containing a month number (1-12, 1=January). d& = long integer containing a day-of-month number (1-31). y& = long integer continaing a year (yyyy).
Remarks:	
Example:	

GetTodaysDate

Procedure

Purpose:	To report elements of the current useful for display.
Author/Reference:	Thomas Wm. Madron (1998)
Syntax:	call GetTodaysDate (WkDay$,Dm$,Mnth$,Year$) or GetTodaysDate WkDay$, Dm$, Mnth$, Year$ DECLARE SUB GetTodaysDate ALIAS "GetTodaysDate" (WkDay$,Dm$,Mnth$,Year$) EXPORT
Parameters:	Parameters are returned by sub GetTodaysDate. WkDay$ - Weekday name. Dm$ - Month number with suffix (i.e., 10th). Mnth$ - Month name. Year$ - Four digit year.
Requires:	FUNCTION zeller FUNCTION LongDayName FUNCTION CompDate FUNCTION CompDayOfMonth FUNCTION LongMonthName FUNCTION CompDayNumber
Remarks:	
Example:	

Today

Function

Purpose:	To report the current date in standard English.
Author/Reference:	Thomas Wm. Madron (1998)
Syntax:	t$ = Today
	DECLARE FUNCTION Today LIB "EWCDATE.DLL" ALIAS "Today" () AS STRING
Parameters:	None.
Requires:	FUNCTION GetTodaysDate
Remarks:	Returns a string formatted as follows: Today is Sunday, June 27th, 1999
Example:	

StandardDate

Function

Purpose:	Function to determine an RFC 822 standard date header line and related functions and subprograms. RFC 822 requires the following date/time format Date: 26 Aug 76 1429 EDT (day, month, year, time, timezone).
Author/Reference:	Thomas Wm. Madron (1998)
Syntax:	d$ = StandardDate DECLARE function StandardDate ALIAS "StandardDate" () export as string
Parameters:	None.
Requires:	FUNCTION CurrentDate FUNCTION ShortMonthName FUNCTION FirstSunday
Remarks:	
Example:	

FirstSunday

Function

Purpose:	To determine the date of the first Sunday of any given month/year.
Author/Reference:	Thomas Wm. Madron (1998)
Syntax:	f& = FirstSunday(month&,year&) DECLARE FUNCTION FirstSunday ALIAS "FirstSunday" (month&,year&) export AS LONG
Parameters:	Month& = long integer containing a month number (1-12, 1=January). Year& = long integer continaing a year (yyyy).
Requires:	FUNCTION zeller
Remarks:	On input, Month and Year must contain the month and year of interest. Day will be returned as the date of the first sunday of the month and year of interest. The first Sunday will, of course, occur sometime within the first seven days of the MONTH.
Example:	

AnyDay

Function

Purpose: To determine the date for any Nth day in a given month/year.

Author/Reference: Thomas Wm. Madron (1998)

Syntax:
```
f& = AnyDay(month&, year&, seq&, wkdy&)

DECLARE FUNCTION AnyDay ALIAS "AnyDay" (month&, year&, seq&,
wkdy&) export AS LONG
```

Parameters:
Month& = long integer containing a month number (1-12, 1=January).
Year& = long integer containing a year (yyyy).
seq& = long integer containing the sequence number of the date sought (i.e., [3]rd Tuesday, [2]nd Wednesday, etc.).
wkdy& = long integer containing the day-of-week number for the day of interest (1-7, 1=Sunday).

Requires:
FUNCTION zeller
FUNCTION DaysInMonth

Remarks: On input, Month and Year must contain the month and year of interest. Day will be returned as the date of the Nth day of the month and year of interest. If the function returns zero (0), then the Nth day was non-existant (i.e., the 5th Monday when there was no 5th Monday).

Example:

WeekDay

Function

Purpose:	GIVEN DAYNUM, FINDS DAY OF WEEK (1=SUN; 2=MON; ETC.)
Author/Reference:	Michael A. Covington, "A Calendar for the Ages," PC TECH JOURNAL, vol. 3, no. 12, Dec. 85, pp. 136ff.
Syntax:	wd& = WeekDay(DN&)
	DECLARE FUNCTION WeekDay LIB "EWCDATE.DLL" ALIAS "WeekDay" (DN&) AS LONG
Parameters:	DN& = long integer containing day number from DayNum.
Requires:	Nothing
Remarks:	
Example:	

CompDayName

Function

Purpose:	Day of the week function.
Author/Reference:	Lewis Rosenfelder, BASIC FASTER AND BETTER (Upland, CA: IJG Inc., 1981), pp. 109-10.
Syntax:	d$ = CompDayName(cmpdt&)
	DECLARE FUNCTION CompDayName ALIAS "CompDayName" (cmpdt&) export as string
Parameters:	cmpdt& - Computational date calculated with CompDate.
Requires:	Nothing.
Remarks:	
Example:	

MonthFromName

Function

Purpose:	To produce a month number from the month's English name.
Author/Reference:	Thomas Wm. Madron (1999)
Syntax:	d& = MonthFromName(Month$)
	DECLARE FUNCTION MonthFromName ALIAS "MonthFromName" (Month$) EXPORT AS LONG
Parameters:	Month$ - Month name (i.e., "January" or "JAN" or "Jan")
Requires:	Nothing.
Remarks:	
Example:	

ParseEnglishDate

Procedure

Purpose:	To parse a date as commonly expressed in English in either U.S. or European formats.
Author/Reference:	Thomas Wm. Madron (1999)
Syntax:	CALL ParseEnglishDate (datestr$, month&, day&, year&, Ecode&) Or ParseEnglishDate datestr$, month&, day&, year&, Ecode& DECLARE SUB ParseEnglishDate ALIAS "ParseEnglishDate" (datestr$, m&, d&, y&, Ecode&) EXPORT
Parameters:	datestr$ - Date string (i.e., October 4, 1937, or 4 October 1937) m& - returned month number. d& - returned day (in month) number. y& - returned year. Ecode& - Number of elements in datestr$ (must = 3 or is invalid).
Requires:	StrTok - local to this dll. MonthFromName& - See documentation.
Remarks:	Parses a date string in either US or European format: US = MonthName DayNumber, Year (4 digits); or European = DayNumber MonthName Year (4 digits). If datestr$ is properly parsed, then Ecode& = 2, else this is not a properly fromatted date string. Ecode& is actually returned as the number of elements in datestr$. If the procedure failes, Ecode& < > 3 (the required number of elements), and m&, d&, and y& are returned as zero (0), otherwise, m&, d&, and y& are returned as the appropriate numbers. Ecode& is set to zero if the parse fails. If either m& or d& evaluate to zero, then Ecode& is returned as zero.
Example:	

Daylight Savings

Daylight saving time is the time during which clocks are set one hour or more ahead of standard time to provide more daylight at the end of the working day during late spring, summer, and early fall. Daylight saving time, also called SUMMER TIME, is a system for uniformly advancing clocks, especially in summer, so as to extend daylight hours during conventional waking time. In the Northern Hemisphere, clocks are usually set ahead one hour in late March or in April and are set back one hour in late September or in October.

The practice was first suggested in a whimsical essay by Benjamin Franklin in 1784. In 1907 an Englishman, William Willett, campaigned for setting the clock ahead by 80 minutes in four moves of 20 minutes each during the spring and summer months. In 1908 the House of Commons rejected a bill to advance the clock by one hour in the spring and return to Greenwich Mean (standard) Time in the autumn.

Several countries, including Australia, Great Britain, Germany, and the United States, adopted summer daylight saving time during World War I to conserve fuel by reducing the need for artificial light. During World War II, clocks were kept continuously advanced by an hour in some nations--e.g., in the United States from Feb. 9, 1942, to Sept. 30, 1945; and England used "double summer time" during part of the year, advancing clocks two hours from the standard time during the summer and one hour during the winter months.

In the United States, daylight saving time formerly began on the last Sunday in April and ended on the last Sunday in October. In 1986 the U.S. Congress passed a law moving up the start of daylight saving time to the first Sunday in April, while keeping its end date the same. In most of the countries of western Europe, daylight saving time starts on the last Sunday in March and ends on the last Sunday in September. In Britain and many other countries worldwide, it lasts from March 30 to October 26. The material concerning the history and practice of daylight saving time is taken from "Daylight saving time." *Britannica CD*. Version 97. Encyclopaedia Britannica, Inc., 1997.

When dealing with dates it is sometimes useful to know whether a given date is part of daylight savings. The general problem with using daylight savings times is that the specific start and stop dates for daylight savings are set legislatively by each nation. The functions dealing with daylight saving time are based on the current (June 27, 1999) U.S. standard. The functions involved are:

FUNCTION DaySavStrt
FUNCTION DaySavEnd
FUNCTION DayLightSavings

DaySavStrt

Function

Purpose:	To calculate the day of the year that Daylight Savings starts (in the US, the 1st Sunday of April, 2:00 a.m.).
Author/Reference:	Thomas Wm. Madron (1998)
Syntax:	ds$ = DaySavStrt(year&)
	DECLARE FUNCTION DaySavStrt ALIAS "DaySavStrt" (year&) export AS LONG
Parameters:	year& = long integer containing a four digit year of interst.
Requires:	Nothing.
Remarks:	
Example:	

DaySavEnd

Function

Purpose:	To calculate the day of the year that Daylight Savings ends (in the US, the last Sunday of October, 2:00 a.m.).
Author/Reference:	Thomas Wm. Madron (1998)
Syntax:	de$ = DaySavEnd(year&)
	DECLARE FUNCTION DaySavEnd ALIAS "DaySavEnd" (year&) export AS LONG
Parameters:	year& = long integer containing a four digit year of interst.
Requires:	Nothing.
Remarks:	
Example:	

DayLightSavings

Procedure

Purpose:	To determine weather a given date falls within the U.S. daylight saving period.
Author/Reference:	Thomas Wm. Madron (1998)
Syntax:	ds$ = DayLightSavings(month&,day&,year&) DECLARE FUNCTION DayLightSavings ALIAS "DaylightSaveings" (month&,day&,year&) EXPORT AS LONG
Parameters:	month& - long integer containing the month of interest. day& - long integer containing the day of interest (within month&). year& = long integer containing a four digit year of interest.
Requires:	FUNCTION DaySavStrt FUNCTION DaySavEnd
Remarks:	Return value: True (-1) if date in daylight savings, else False (0).
Example:	

Local Utility Functions

The following utility functions are used, at this point, only to support other routines and are not accessible from external programs. They are private to this DLL and cannot be accessed by other programs.

Support for the Covington collection:

FUNCTION rgtALIGN$
FUNCTION DFrac
FUNCTION Dint
FUNCTION Floor

Support for the Madron collection:

SUB StrTok

rtALIGN

Function

Purpose: To right align text in a given length of screen space.

Author/Reference: Michael A. Covington, "A Calendar for the Ages," PC TECH JOURNAL, vol. 3, no. 12, Dec. 85, pp. 136ff.

Syntax:
```
text$ = rgtALIGN(text$,length&)

DECLARE FUNCTION rgtALIGN$ ALIAS "rgtALIGN" (text$,length&)
```

Parameters: text$ - any arbitrary text.
length& - length of the field into which text is being right aligned.

Requires: Nothing.

Remarks:

Example:

DFrac

Function

Purpose:	To return the fractional part of a Real number.
Author/Reference:	Michael A. Covington, "A Calendar for the Ages," PC TECH JOURNAL, vol. 3, no. 12, Dec. 85, pp. 136ff.
Syntax:	Y# = Dfrac(X#)
	DECLARE FUNCTION DFrac ALIAS "DFrac" (X#) AS DOUBLE
Parameters:	X# = a double precision real number from which the fractional part is to be extracted.
Requires:	Nothing.
Remarks:	
Example:	

DInt

Function

Purpose:	To truncate to Double Precision.
Author/Reference:	Michael A. Covington, "A Calendar for the Ages," PC TECH JOURNAL, vol. 3, no. 12, Dec. 85, pp. 136ff.
Syntax:	Y# = Dint(X#)
	DECLARE FUNCTION Dint ALIAS "Dint" (x#) AS DOUBLE
Parameters:	X# = a double precision real number
Requires:	Nothing.
Remarks:	
Example:	

Floor

Function

Purpose:	Find the largest whole number not greater than X# (a real number).
Author/Reference:	Michael A. Covington, "A Calendar for the Ages," PC TECH JOURNAL, vol. 3, no. 12, Dec. 85, pp. 136ff.
Syntax:	Y# = Floor(X#)
	DECLARE FUNCTION Floor ALIAS "Floor" (X#) AS DOUBLE
Parameters:	X# = double precision real number.
Requires:	Nothing.
Remarks:	
Example:	

StrTok

Procedure

Purpose:	To parse a list of tokens and return them in Params$(i).
Author/Reference:	Thomas Wm. Madron (1998)
Syntax:	StrTok Params$(), CountParams&, Delimiter$, Tokens$ Or, CALL StrTok (Params$(), CountParams&, Delimiter$, Tokens$) DECLARE SUB StrTok ALIAS "StrTok" (Params$(), CountParams&, Delimiter$, Tokens$, Ecode&)
Parameters:	Params$() - a string array Dimensioned to the largest number of tokens that might be encountered. CountParams& - long integer returned as the number of tokens found. Delimiter$ - the character(s) separating each token. Tokens$ - a string containing the tokens to be parsed. Ecode& - error code: 0=OK; -1=No Delimiter
Requires:	Nothing.
Remarks:	Similar in purpose to the C function StrTok, this procedure differs only in the fact that all tokens are returned at once in an array rather than one at a time. The number of tokens is returned in CountParams&. On input, a single Delimiter$ must be specified or an error code (-1) will be returned; and one or more Tokens must be contained in Token$, separated by Delimiter$. Spaces can also be used independently of Delimiter$ to make the input more readable.
Example:	

Sample Program
TSTDT.BAS

A small and simple sample program that exercises all the functions and sub programs described above is the following:

---------- Cut Here ----------

```
$INCLUDE "ewcdate.inc"

DECLARE FUNCTION padright$ (text$, padchar$, fieldlen%)
DECLARE FUNCTION edtnum$ (n&)

FUNCTION pbmain () AS LONG
      CLS
      stdout padright$("Function/Procedure"," ",29)+"Result"
      stdout padright$("","=",28)+" "+padright$("","=",35)
      CALL CurrentDate (m&, d&, y&):
      stdout padright$("CurrentDate"," ",29)+edtnum$(m&)+" "+edtnum$(d&)+" "+edtnum$(y&)
      stdout padright$("Long|ShortDayName"," ",29)+"Today is "+LongDayName(DayOfWeek(m&,d&,y&))+" or
            "+ShortDayName (DayOfWeek(m&,d&,y&))
      stdout padright$("Long|ShortMonthName"," ",29)+"This month is "+LongMonthName (m&)+" or
            "+ShortMonthName (m&)
      if DayLightSavings (m&, d&, y&) then
            stdout padright$("DayLightSavings"," ",29)+"Time period is daylight savings."
      else
            stdout padright$("DayLightSavings"," ",29)+"Time period is standard."
      end if
      CALL GetTodaysDate (WkDay$,Dm$,Mnth$,Year$)
      stdout padright$("GetTodaysDate"," ",29)+WkDay$+" "+Dm$+" "+Mnth$+" "+Year$
      stdout padright$("Today"," ",29)+Today
      stdout padright$("Julian"," ",29)+edtnum$(julian (m&, d&, y&))
      CALL JulToDate (Julian(m&,d&,y&), ResultDate$)
      stdout padright$("JulToDate"," ",29)+ResultDate$
      stdout padright$("DayOfYear"," ",29)+edtnum$(DayOfYear (m&, d&, y&))
      stdout padright$("DayOfWeek"," ",29)+edtnum$(DayOfWeek (m&, d&, y&))
      stdout padright$("StandardDate"," ",29)+StandardDate
      stdout padright$("Zeller"," ",29)+edtnum$(zeller (m&,d&,y&))
```

```
      stdout padright$("FirstSunday"," ",29)+edtnum$(FirstSunday (m&,y&))
      stdout padright$("DaysInMonth"," ",29)+edtnum$(DaysInMonth (m&, y&))
      tmp& = LeapYear(y&)
      select case tmp&
            case 0
                  ans$ = "No"
            case -1
                  ans$ = "Yes"
      end select
      stdout padright$("LeapYear"," ",29)+"("+edtnum$(tmp&)+") "+ans$
      dn& = DayNum (m&, d&, y&)
      stdout padright$("DayNum"," ",29)+edtnum$(dn&)
      call CALDAT (dn&, Month&, Day&, Year&)
      stdout padright$("CalDate"," ",29)+edtnum$(Month&)+" "+edtnum$(Day&)+" "+edtnum$(Year&)
      stdout padright$("WeekDay"," ",29)+edtnum$(WeekDay (dn&))
      stdout padright$("JulianDate"," ",29)+edtnum$(JulianDate (m&, d&, y&))
      cmpdt& = CompDate (m&, d&, y&)
      stdout padright$("CompDate"," ",29)+edtnum$(cmpdt&)
      stdout padright$("CompYear"," ",29)+edtnum$(CompYear (cmpdt&))
      stdout padright$("CompDayNumber"," ",29)+edtnum$(CompDayNumber (cmpdt&))
      stdout padright$("CompDayName"," ",29)+CompDayName (cmpdt&)
      stdout padright$("CompDayOfMonth"," ",29)+edtnum$(CompDayOfMonth (y&, m&, CompDayNumber (cmpdt&)))
      stdout padright$("AnyDay"," ",29)+edtnum$(AnyDay (m&, y&, 3, 3))+" (third tuesday)"
      stdout padright$("MonthFromName"," ",29)+edtnum$(MonthFromName(LongMonthName(m&)))
      datestr$ = LongMonthName(m&)+" "+edtnum$(d&)+", "+edtnum$(y&)
      ParseEnglishDate datestr$, month&, day&, year&, Ecode&
      stdout padright$("ParseEnglishDate"," ",29)+"("+datestr$+") = "+edtnum$(month&)+" "+edtnum$(day&)+"
            "+edtnum$(year&)
      stdout padright$("","=",64)
      stdout "This test program executes all functions and subprograms in"
      stdout "ewcdate.dll.  The resultsof this program can be redirected"
      stdout "to a file for more liesurely inspection:  tstdt > filename.txt."
      stdout "The output may also be paged using more, or if you have it,"
      stdout "less:  tstdt | more, or tstdt | less."
      stdout padright$("","=",64)
END FUNCTION

FUNCTION padright$ (text$, padchar$, fieldlen%)
    LOCAL i%
```

```
    i% = LEN(text$)
    SELECT CASE i%
         CASE > fieldlen%
              padright$ = LEFT$(text$, fieldlen%)
         CASE = fieldlen%
              padright$ = text$
         CASE < fieldlen%
              padright$ = rtrim$(text$) + STRING$(fieldlen%-LEN(text$),padchar$)
         CASE ELSE
              EXIT SELECT
    END SELECT
END FUNCTION

FUNCTION edtnum$ (n&)
    '
    '   Present an integer number as a string without leading spaces.
         This could, of course, be replaced with the PBDLL function trim$.
    '
        edtnum$ = ltrim$(rtrim$(str$(n&)))
END FUNCTION
```

Validity Testing

test.bas

This program is part of the distribution and verifies the adequacy of those functions and procedures using Julian-like numbers .

```
'   Test the accuracy of date to julian to date functions
'
$INCLUDE "j:\compiler\inc\ewcdate.inc"

FUNCTION PBMAIN () AS LONG
    CLS
    RANDOMIZE
    StartYear& = 1699
    EndYear& = 2099
    STDOUT "Julian and Computational Date Validation"
    STDOUT ""
    PRINT "Input","Source","Julian#","Result"
    PRINT "==========","==========","==========","=========="
    FOR Year& = StartYear& TO EndYear& STEP 100
        ' Grillo:
        month& = RND(1,12)
        ndays& = DaysInMonth (month&, year&)
        Day& = RND(1, ndays&)
        TestDate$ = TRIM$(STR$(Month&))+"/"+TRIM$(STR$(Day&))+"/"+TRIM$(STR$(Year&))
        grillo# = julian (Month&, Day&, Year&)
        CALL JulToDate (grillo#, ResultDate$)
        PRINT TestDate$, "Grillo", STR$(grillo#), ResultDate$
    NEXT year&
    STDOUT ""

    FOR year& = StartYear& TO EndYear& STEP 100
        ' Covington
        month& = RND(1,12)
        ndays& = DaysInMonth (month&, year&)
        Day& = RND(1, ndays&)
        TestDate$ = TRIM$(STR$(Month&))+"/"+TRIM$(STR$(Day&))+"/"+TRIM$(STR$(Year&))
        covington& = DayNum (Month&, Day&, Year&)
        CALL CalDat (covington&, m&, d&, y&)
```

```
        PRINT TestDate$, "Covington", STR$(covington&), _
            TRIM$(STR$(m&))+"/"+TRIM$(STR$(d&))+"/"+TRIM$(STR$(y&))
    NEXT year&

    STDOUT ""
    FOR year& = StartYear& TO EndYear& STEP 100
        ' Rosenfelder
        month& = RND(1,12)
        ndays& = DaysInMonth (month&, year&)
        Day& = RND(1, ndays&)
        TestDate$ = TRIM$(STR$(Month&))+"/"+TRIM$(STR$(Day&))+"/"+TRIM$(STR$(Year&))
        rosenfelder& = CompDate (month&, day&, year&)
        y& = CompYear(rosenfelder&)
        m& = CompMonth (rosenfelder&, y&)
        cmpdaynum& = CompDayNumber (rosenfelder&)
        d& = CompDayOfMonth (y&, m&, cmpdaynum&)
        PRINT TestDate$, "Rosenfelder", STR$(rosenfelder&), _
            TRIM$(STR$(m&))+"/"+TRIM$(STR$(d&))+"/"+TRIM$(STR$(y&))
    NEXT year&

END FUNCTION
```

Bibliography

Michael A. Covington, "A Calendar for the Ages," *PC TECH JOURNAL*, vol. 3, no. 12, Dec. 85, pp. 136ff.

John P. Grillo, and J. D. Robertson, *SUBROUTINE SANDWICH* (New York: John Wiley & Sons, 1983).

John P. Grillo and J. D. Robertson, *MORE SUBROUTINE SANDWICH* (New York: John Wiley & Sons, 1983).

Jean Meeus, *ASTRONOMICAL FORMULAE FOR CALCULATORS*, 2nd Edition (Richmond, VA: Willmann-Bell, 1982).

J. D. Robertson, "Remark on Algorithm 398," *COMMUNICATIONS OF THE ACM*, Vol. 15, No. 10, 1972, p. 918.

Lewis Rosenfelder, *BASIC FASTER AND BETTER* (Upland, CA: IJG Inc., 1981).

"Julian Period." *Britannica CD*. Version 97. Encyclopaedia Britannica, Inc., 1997.

"Daylight saving time." *Britannica CD*. Version 97. Encyclopaedia Britannica, Inc., 1997.