Date Validation and Conversion Utility
API Reference
Version 1.0
October 26, 1991

Written By
Albert C. Schmitt
CIS 70541,2362

Notice

Credits

The mathematical formulae that convert a date to and from its Julian representation are derived in part from the Microsoft QuickC Programmer's Toolbox by John Clark Craig published by Microsoft Press.

Installation

The Date Conversion and Validation Utility consists of the files:

DATEVLD.DLL - The date utility.
DATEVLD.H - The 'C' header file.
DATEVLD.LIB - The compile time link library.
DATEVLD.PAS - The TPW interface unit.
DATEVLD.WRI - This document.
EXAMPLE.ZIP - Test programs (Borland C, QuickC Win & TPW).

C users should copy the DATEVLD.DLL to your WINDOWS directory or any directory that is in your path statement. Copy DATEVLD.H to your compiler's include directory and DATEVLD.LIB to your compiler's library directory.

TPW users should copy DATEVLD.DLL to your WINDOWS directory and DATEVLD.PAS to a directory in your TPW directory path.

Introduction

The functions described in this reference are designed to allow you to easily incorporate date validation into your Windows 3.0 'C', 'C++' or 'Turbo Pascal for Windows' application. This utility may also be incorporated into any language that supports Windows 3.0 Dynamic Link Libraries, such as Visual Basic (I leave this up to you).

Overview

This library was designed to allow you, the developer, to quickly and easily add date validation to your application. In addition, this library automatically recognizes the date format as specified in the International section of Control Panel. This feature gives control of the format in which the date is displayed and entered to the user.

Throughout this document, the term *Julian date* is refered to. The Julian date is a long integer representing the number of days from October 15, 1582. Since October 16, 1582 was the first day of our present *Gregorian* calendar, its Julian value would be 1. The date October 27, 1991 is 149395 Julian or 149395 days after 10/16/1582. Although this library contains 10 functions, just 2 of them allow you to perform nearly all of the validations and conversions that you you might require. All dates from October 16, 1582 through December 31, 9999 are valid.

Upon startup, the utility reads the current date format as defined in the [Intl] section of WIN.INI. These values

are set from the Control Panel "International" Icon. Windows supports 3 date formats: MDY, DMY, or YMD. In addition 2 or 4 digit year formats and leading zeros may be added to the month and/or day. This allows the following formatting options:

MDY formats	**DMY formats**	**YMD formats**
MM/dd/yy	dd/MM/yy	yy/MM/dd
MM/dd/yyyy	dd/MM/yyyy	yyyy/MM/dd
MM/d/yy	d/MM/yy	yy/MM/d
MM/d/yyyy	d/MM/d/yyyy	yyyy/MM/d
M/dd/yy	dd/M/yy	yy/M/dd
M/dd/yyyy	dd/M/yyyy	yyyy/M/dd
M/d/yy	d/M/yy	yy/M/d
M/d/yyyy	d/M/yyyy	yyyy/M/d

If the [Intl] section of WIN.INI is missing MM/dd/yy is assumed. In addition to allowing each user to control his or her own date format, DATEVLD allows users to use any non-numeric value as a separator, such as a space. For example, if MM/dd/yyyy is the date format and "1 3 58" is entered DATEVLD would interpret this as "01/03/1958". Since this is so, it would be wise to immediately reformat and display the date so that it will appear formatted properly. This technique is discussed in the next section.

Examples

To verify that a user entered the correct date you would perform the following steps:

1. Get the date from the edit field into a string buffer.
2. Call the **toJul** function, passing it the string date.
3. Check the value returned from **toJul**. If it is not zero (0L) then the date is valid. If it is zero then the user entered an invalid date or the date did not conform to the format in the International section of Control Panel.
4. If the date is invalid then display an appropriate message and repeat step 1.

The following code fragment illustrates the above procedure:

```
LONG jDate;
char datebuf[12];

GetDlgItemText(hDlg, IDM_DATE, (LPSTR)datebuf, 11);
jDate = toJul((LPSTR)datebuf);
if (jDate == 0L)
      MessageBox(hDlg, "Invalid Date", "Generic", MB_OK |
MB_ICONEXCLAMATION);
```

The **toDate** function converts a Julian date into a string date of the format specified in the International section of Control Panel. This is useful when you want to increment or decrement the date or reformat a date after it is entered by the user. You may want to do this if the date is valid since the **toJul** function recognizes any non-numeric values as date separators. For example, the value "10 4 91" might be entered. Since this is a valid date, converting its Julian value back to a string will result in the more appropriate "10/04/1991". The following code fragment illustrates this:

```
LONG jDate;
char datebuf[12];
```

```
GetDlgItemText(hDlg, IDM_DATE, (LPSTR)datebuf, 11);
jDate = toJul((LPSTR)datebuf);
if (jDate != 0L)
{
      toDate((LPSTR)datebuf, jDate);
      SetDlgItemtext(hDlg, IDM_DATE, (LPSTR)datebuf);
} else
      MessageBox(hDlg, "Invalid Date", "Generic", MB_OK |
MB_ICONEXCLAMATION);
```

To increment the the date entered by the user, convert it to a Julian, increment it, then convert it back to a string. The following code fragment illustrates how to do this:

```
jDate = toJul(datebuf);       /* assumes datebuf contains a valid date */
toDate(datebuf, jDate + 1L);
```

Another function, **StrDate**, accepts an integer month, day and year and returns a properly formatted date string. You should note that **StrDate** makes no attempt whatsoever to valdate the date sent to it. The following code fragment illustrates how to format and display the system date using Borland C++:

```
struct date d;
char datebuf[12];

getdate(&d);
StrDate((LPSTR)datebuf, d.da_mon, d.da_day, d.da_year);
SetDlgItemText(hDlg, IDM_DATE, (LPSTR)datebuf);
```

To calculate the number of days between any two dates do the following:

```
LONG beginDate, endDate, numDays;

endDate = toJul(lpszEndDate);
beginDate = toJul(lpszBeginDate);
numDays = endDate - beginDate;
```

In the above example, lpszEndDate and lpszBeginDate represent long pointers to zero terminated date strings which are converted to Juilan values and subtracted. If numDays is < 0 then the end date was earlier than the begin date. For purposes of illustration, this code fragment assumes that the string dates are valid. In an actual program, you would explicitly test for valid Julian dates before performing the calculation.

Summary

The above examples should get you started. If you create an interface for a language other than those provided, please send it to me so that I can include it in the next release.

If you have any questions or comments (or discover any bugs) contact me via my Compuserve address 70541,2362. Enjoy.

---

dayName

Syntax **LPSTR FAR PASCAL dayName**(jdate)

This function returns a string containing the day of the week for the given Julian date. The jdate value is converted into an integer day of week which is then used to subscript into an array of day names.

Parameter	Type/Description
jdate	**LONG** Specifies the Julian date.

Return Value A constant string containing the day of week: "Friday", "Saturday", "Sunday," "Monday", "Tuesday", "Thursday".

Comments This function should not be used where international support is desired since the day names are English. Instead, use the **dayOfWeek** function and a string table in your resource file which contains the day names in the desired language.

---

dayOfWeek

Syntax **int FAR PASCAL dayOfWeek**(jdate)

This function converts the given Julian date to an integer day of the week.

Parameter	Type/Description
jdate	**LONG** Specifies the Julian date.

Return Value An integer corresponding to the day of the week 0=Friday, 1=Saturday, 2=Sunday, 3=Monday, 4=Tuesday, 5=Wednesday, 6=Thursday.

---

daysInMonth

Syntax **int FAR PASCAL daysInMonth**(month, year)

This function returns the number of days in the month of the given year. The year is used to calculate the number of days in February for leap years and leap centuries.

Parameter	Type/Description
month	**int** Identifies the month (1 to 12) where 1=January, 12=December.
year	**int** Identifies the year (1582 to 9999). If year is 0 to 99 then 1900 to 1999 is assumed.

Return Value The number of days in the month.

---

isLeapYear

Syntax **int FAR PASCAL isLeapYear**(year)

This function returns TRUE if year is a leap year, FALSE if not . If year is 0 to 99 then 1900 to 1999 is assumed. Leap centuries are also considered by this function.

Parameter	Type/Description
year	**int** The year.

Return Value TRUE if the given year is a leap year, False if it is not.

StrDate

Syntax **VOID FAR PASCAL StrDate** (strDate, month, day, year)

This function converts the given month, day and year into a string. The string date is formatted as specifed in the International section of Control Panel.

Parameter	Type/Description
strDate	**LPSTR** Points to the buffer which will receive the formatted date string.
month	**int** The month.
day	**int** The day.
year	**int** The year.

Return Value None.

Comments This function does not verify that the given month, day and year are valid. The strDate parameter must be long enough to hold the date string.

---

toDate

Syntax **VOID FAR PASCAL toDate**(sdate, jdate)

This function converts the given Julian date into a string. The string date is formatted as specifed in the International section of Control Panel.

Parameter	Type/Description
sDate	**LPSTR** Points to the buffer which will receive the formatted date string.
jdate	**LONG** The Julian date.

Return Value None.

Comments The sdate parameter must be long enough to hold the date string.

---

toJul

Syntax **LONG FAR PASCAL toJul** (sdate)

This function converts the given string date to its Julian value. The string date must be formatted as specifed in the International section of Control Panel.

Parameter	**Type/Description**
sDate	**LPSTR** Points to the buffer which contains the date string.
jdate	**LONG** The Julian date.

Return Value The Julian date.

Comments This function recognizes any non-numeric values as valid month, day, year separators.

---

yearName

Syntax **int FAR PASCAL yearName**(LONG jdate)

This function returns the integer year from the given Julian date.

Parameter	**Type/Description**
jdate	**LONG** The Julian date.

Return Value An integer year in the range 1582 to 9999.

_fromJulian

Syntax **LONG FAR PASCAL _fromJulian** (month, day, year, jdate)

This function returns the month, day and year for the given Julian date.

Parameter	**Type/Description**
month	**LPINT** Points to an integer which will contain the month.
day	**LPINT** Points to an integer which will contain the day.
year	**LPINT** Points to an integer which will contain the year.
jdate	**LONG** The Julian date to be be converted.

Return Value The Julian date that was sent to this function. Fills in the month, day and year parameters for the given Julian date.

_toJulian

Syntax **LONG FAR PASCAL _toJulian** (month, day, year)

This function converts the given month, day and year to their Julian value.

Parameter	Type/Description
month	**int** The month.
day	**int** The day.
year	**int** The year.

Return Value The Julian date or 0L if the month, day, year are invalid.

Comments This function verifies that the given month, day and year are valid.