CHAPTER 6

Using DATE.PL

OVERVIEW

Although PERL has many built-in routines to do a fantastic job of parsing files and manipulating strings, it does not excel in the area of performing functions on dates. This chapter will add the **DATE.PL** library to our programming arsenal. **DATE.PL** was originally written by Gary Puckering.

DATE.PL works by converting dates to and from Julian dates. The Julian date system basically converts dates to a single whole integer. This is important because it allows us to easily calculate how many days exist between two calendar dates without worrying about such things as leap years and how many days are in each month; **DATE.PL** takes care of all that for you.

Chapter 13 uses this library to perform feats of date arithmetic without resorting to calling an operating system–specific function such as the `date` or `cal` command on UNIX. Although `date` and `cal` can return some useful information, a good CGI programmer should stay away from using operating system specific calls since it lessens the portability of the application to other platforms.

The **DATE.PL** library contains main functions to convert a Julian date to a month, day, year, and weekday and to do the reverse. In addition, DATE.PL has several functions that allow you to convert month names to month numbers, month numbers to month names, and weekday numbers to weekday names. Other functions get the Julian date for today, tomorrow, and yesterday.

INSTALLATION AND USAGE

Converting Julian Dates

The `jday` function takes a month, day, and year as parameters and returns a single integer that is a Julian date. Similarly, the `jdate` function returns the month, day, year, and weekday based on a Julian date passed to it as a parameter. Simple Perl code appears below, which uses these functions. The first line gets the Julian date of "12/1/95." The second line gets the correct date parameters (month, day, year, weekday) from the "1000" Julian date.

```
$juliandate=& jday (12,1,95)'
($month, $day, $yearn $weedat)=& jdate(1000);
```

Converting Month Numbers

The `monthname` function uses a month number parameter and returns a descriptive name of the month. The `monthnum` function takes three or more letters of the month name and returns the corresponding month number. For example, January is month number 1. The following code would print `1 : January`. Note that the numbers passed to the monthname function do not have to be in any special format. You can pass "01," "1," or even "1.0" and monthname will return "January" for all of them.

```
print &monthnum("JAN") . ":" . &monthname(1);
```

Converting Weekday Numbers

The `weekday` function takes a weekday number, where 0 is Sunday, and returns an abbreviated weekday name. For example, if `$number` equals 2, the following code would produce `Tue` as the output: Just like monthname, the weekday routine accepts any whole integer as a parameter regardless of how it is formatted.

```
print &weekday($number);
```

Obtaining the Julian Date for Yesterday, Today, and Tomorrow

The `today`, `yesterday`, and `tomorrow` functions return the Julian dates for, respectively, today, yesterday, and tomorrow.

Design Discussion

Main Script

The first line of the **DATE.PL** library defines the whole file as part of a `package` called `date`. Normally, when a variable is declared outside a function, it is declared as global and so it affects all other functions in the currently running Perl program. Using a `package` allows declaration of variables that have a scope only within the `package` but yet are relevant outside the scope of the individual functions of the `package`. After this, a base Julian date number is assigned to `$brit_jd` for use in the following algorithms:

```
package date;

# The following defines the first
# day that the Gregorian calendar was used
# in the British Empire (Sep 14, 1752).
# The previous day was Sep 2, 1752
```

```
# by the Julian calendar.  The year began
# at March 25th before this date.

$brit_jd = 2361222;
```

jdate Subroutine

The `jdate` function takes a Julian date and converts it back to a normal month, day, year, and weekday number. The `WARN` command, used in the following code to warn of a problem in the date range, acts just like `DIE` in that the message it is passed prints to `STDERR`, but it does not exit the program. The rest of the function uses a standard algorithm for converting the Julian date back into our normal date format information.

```
sub main'jdate
# Usage:  ($month,$day,$year,$weekday) = &jdate($julian_day)
{
     local($jd) = @_;
     local($jdate_tmp);
     local($m,$d,$y,$wkday);

     warn("warning:  pre-dates British use of Gregorian calendar\n")
          if ($jd < $brit_jd);

     # calculate weekday (0=Sun,6=Sat)
     $wkday = ($jd + 1) % 7;
     $jdate_tmp = $jd - 1721119;
     $y = int((4 * $jdate_tmp - 1)/146097);
     $jdate_tmp = 4 * $jdate_tmp - 1 - 146097 * $y;
     $d = int($jdate_tmp/4);
     $jdate_tmp = int((4 * $d + 3)/1461);
     $d = 4 * $d + 3 - 1461 * $jdate_tmp;
     $d = int(($d + 4)/4);
     $m = int((5 * $d - 3)/153);
     $d = 5 * $d - 3 - 153 * $m;
     $d = int(($d + 5) / 5);
     $y = 100 * $y + $jdate_tmp;
     if($m < 10) {
          $m += 3;
     } else {
          $m -= 9;
          ++$y;
```

```
     }
     ($m, $d, $y, $wkday);
}
```

jday Subroutine

The `jday` function converts a normal date to a numerical Julian date. Note that the year must be a four-digit year. The divisions by 4 on the year in the following code are based on the fact that every four years is a leap year. The `WARN` function is used here to indicate a potential problem with the Julian date. The rest of the program is the actual mathematical algorithm for converting to a Julian day.

```
sub main'jday
# Usage:  $julian_day = &jday($month,$day,$year)
{
     local($m,$d,$y) = @_;
     local($ya,$c);

     $y = (localtime(time))[5] + 1900  if ($y eq '');

     if ($m > 2) {
          $m -= 3;
     } else {
          $m += 9;
          --$y;
     }
     $c = int($y/100);
     $ya = $y - (100 * $c);
     $jd =  int((146097 * $c) / 4) +
             int((1461 * $ya) / 4) +
             int((153 * $m + 2) / 5) +
             $d + 1721119;
     warn("warning:  pre-dates British use of Gregorian calendar\n")
          if ($jd < $brit_jd);
     $jd;
}
```

is_jday Subroutine

The `is_jday` function returns true if the value passed to it is within a reasonable date range of what would be considered a Julian date:

```
sub main'is_jday
{
# Usage:  if (&is_jday($number)) { print "yep - looks like a jday"; }
     local($is_jday) = 0;
     $is_jday = 1 if ($_[0] > 1721119);
}
```

monthname Subroutine

The `monthname` function takes the month number and returns the month name. It does this by taking the month number and using it as an index to an array of month names. If a second parameter is also passed to the `monthname` function, the month name that is returned will be truncated to the number of characters specified by the second parameter.

Note that in the code below, the month number that is passed as a parameter has 1 subtracted from it when the routine references the `@names` array. This is done because although the month numbers go from 1 through 12, arrays in Perl start being indexed at element number 0. Thus, the numbers that reference the month names in the @names array must go from 0 through 11 instead of 1 through 12.

Similar logic is used by Perl to truncate the month name if the $m parameter is specified. Basically, the `substr` command is called with the following parameters: string to extract, offset in the string to start extracting, and the length of the extraction. The similarity to how arrays are handled above is contained in the fact that the offset index into the character string starts at 0 for the first character. Thus, the command `substr($names[$n-1], 0, $m)` returns the month name starting from character 0 (first character) as a length of $m characters.

```
sub main'monthnum

sub main'monthname
# Usage:  $month_name = &monthname($month_no)
{
     local($n,$m) = @_;
     local(@names) =
('January','February','March','April','May','June',

'July','August','September','October','November',
```

```
                    'December');
     if ($m ne '') {
          substr($names[$n-1],0,$m);
     } else {
          $names[$n-1];
     }
}
```

monthnum subroutine

The `monthnum` function takes the month name and converts it to a month number. It does this by taking the month name, stripping it to the first three characters, and making it uppercase. This conversion makes it compatible with an associative array that has the three-character month names as keys to the month number values. The appropriate element in the associative array is then returned in response to the month name.

```
# Usage:  $month_number = &monthnum($month_name)
{
     local($name) = @_;
     local(%names) = (
     'JAN',1,'FEB',2,'MAR',3,'APR',4,'MAY',5,'JUN',6,
          'JUL',7,'AUG',8,
          'SEP',9,'OCT',10,'NOV',11,'DEC',12);
     $name =~ tr/a-z/A-Z/;
     $name = substr($name,0,3);
     $names{$name};
}
```

weekday Subroutine

The `weekday` function takes a weekday number and converts it to a three-character weekday abbreviation. The function works by taking the value and using it as an index to an array of weekday names:

```
sub main'weekday
# Usage:  $weekday_name = &weekday($weekday_number)
{
     local($wd) = @_;
     ("Sun","Mon","Tue","Wed","Thu","Fri","Sat")[$wd];
}
```

today Subroutine

The `today` function returns today's Julian date. First, it calls the `localtime(time)` Perl function in which the elements of the current time and date are broken up as an array. The date, month, and year are extracted from this function as `$d`, `$m`, and `$y`. The month is incremented by 1, because the `localtime` function returns months on a scale from 0 to 11 instead of 1 to 12. The year is incremented by 1900 to provide a four-digit year. Then the `jday` function is called to convert this number to today's Julian date:

```
sub main'today
# Usage:  $today_julian_day = &today()
{
     local(@today) = localtime(time);
     local($d) = $today[3];
     local($m) = $today[4];
     local($y) = $today[5];
     $m += 1;
     $y += 1900;
     &main'jday($m,$d,$y);
}
```

yesterday and tomorrow subroutines

The following functions—`yesterday` and `tomorrow`—operate by calling the `today` function and, respectively, subtracting 1 from or adding 1 to the returned Julian date:

```
sub main'yesterday
# Usage:  $yesterday_julian_day = &yesterday()
{
     &main'today() - 1;
}

sub main'tomorrow
# Usage:  $tomorrow_julian_day = &tomorrow()
{
     &main'today() + 1;
}
```