Dates and Numbers

As you might be able to guess from the name, the packages in this section allow you to deal with dates and numbers. The Date package has an extensive API that allows you to compare dates, add and subtract time spans, convert time zones, and perform just about every other date-related function you can imagine. The Date_Holidays package allows you to see if a given date is a holiday, to list all holidays between two dates, and to find out what the date is for a given holiday. It works with a variety of different holiday *drivers*, including those for Christian, Jewish, US, and United Nations Organization (UNO) holidays, to name a few. Numbers_Roman has a small API, but has the useful purpose of converting given numbers into Roman numerals and converting Roman numerals into numbers. Finally, Numbers_Words is capable of printing the text representation of numbers.

Date

The Date package provides a `Date` class that offers many methods for dealing with dates. The `Date` constructor accepts a string representation of a `Date` and builds an object that can be formatted, added to, subtracted from, and compared to other dates. The Date package includes two other useful classes, `Date_Span` and `Date_TimeZone`, which are also detailed in the API documentation in this chapter. The `Date_Span` class allows you to build time spans easily that can be added or subtracted to and from `Date` values. The `Date_TimeZone` class allows you to get the information about a time zone, including its offset from Coordinated Universal Time (UTC), and to get daylight saving time information using a short name for the time zone.

Common Uses

The Date package is used for the following purposes:

- Formatting dates
- Adding and subtracting dates
- Comparing dates

Related Packages

The following package is related because it depends on this one:

Date_Holidays

Dependencies

The Date package has the following dependencies.

Required Packages

None

Optional Packages

None

Date API

Date Constructor

Creates a new instance of a `Date` object, with the current date and time if no parameter is passed. You can provide a date to the constructor in a couple different formats, including ISO 8601 (for example, `1997-08-29 02:14:00`), timestamp (for example, `19970829021400`), or Unix time (for example, `872820840`) formats. You may also pass in another `Date` object.

```
Date Date ([mixed $date = null])
```

Parameter	Type	Description
$date	mixed	A string representation of a date.

addSeconds()

Adds the given number of seconds to the current date and time, changing the value.

```
void addSeconds(integer $sec)
```

Parameter	Type	Description
$sec	integer	The number of seconds to add to the `Date`.

addSpan()

Adds the given `Date_Span` object (the API of which is detailed later in this chapter) to the current date.

```
void addSpan(Date_Span $span)
```

Parameter	Type	Description
$span	Date_Span	A `Date_Span` object that contains the specification for a unit of time that can be added to the `Date`.

after()

Returns `true` if the `Date` object is chronologically after the given date.

```
boolean after(Date $when)
```

Parameter	Type	Description
$when	Date	Another `Date` object to compare to the `Date`.

before()

Returns `true` if the `Date` object is chronologically before the given date.

```
boolean before(Date $when)
```

Parameter	Type	Description
$when	Date	Another `Date` object to compare to the `Date`.

compare()

Returns a signed integer that indicates the relationship between the two dates. If `$d1` is greater than `$d2`, the method will return `1`. If `$d1` is less than `$d2`, the method will return `-1`. `compare()` returns `0` if the `Date` objects are equal. This is a good method to use for sorting dates.

```
integer compare(Date $d1, Date $d2)
```

Parameter	Type	Description
$d1	Date	One `Date` object in the comparison.
$d2	Date	The other `Date` object in the comparison.

convertTZ()

Converts the `Date`'s time zone to the time zone represented by the `Date_TimeZone`.

```
void convertTZ(Date_TimeZone $tz)
```

Parameter	Type	Description
$tz	Date_TimeZone	An object that represents a time zone.

Caution This method uses a function called `putenv()` to obtain the time zone information, especially when getting information about whether or not the current time zone is in daylight saving time. This method might not work on all operating systems (such as Microsoft Windows) because it relies on underlying calls for the information.

convertTZbyID()

Converts the `Date`'s time zone to the time zone represented by the given string identifier.

```
void convertTZbyID(string $id)
```

Parameter	Type	Description
$id	string	An identifier for a time zone: CST, DST, MST.

■ **Caution** The same limitations with `convertTZ()` apply to `convertTZbyID()`.

copy()

Copies the value of `$date` into the `Date` object.

```
void copy(Date $date)
```

Parameter	Type	Description
$date	Date	Another `Date` from which to copy the values into the `Date` object.

equals()

Returns `true` if the provided `Date` is equal to the instantiated `Date` object.

```
boolean equals(Date $when)
```

Parameter	Type	Description
$when	Date	Another `Date` to compare for determining if the two `Date` objects are equal.

format()

Formats the `Date` into a string that can include many different date parts.

```
string format(string $format)
```

Parameter	Type	Description
$format	string	The format in which the `Date` will be printed.

The following table lists the format string objects, along with what they represent.

Format Strings for Formatting Date Objects

Format	Description	Example
%a	Short weekday name.	Sun, Mon, Tue, Wed
%A	Long weekday name.	Sunday, Monday, Tuesday, Wednesday
%b	Short month name.	Jan, Feb, Mar, Apr
%B	Long month name.	January, February, March, April
%C	Century	18, 19, 20 *(1800s, 1900s, and 2000s, respectively).*
%d	Day of month.	00, 01, 02, 03 . . . 31
%D	Short date format.	03/04/05 (shortcut for %m/%d/%y)
%e	Single digit day of month.	0, 1, 2, 3 . . . 31
%E	Number of days since epoch.	3223, 23333, 34333
%H	Hour as number.	00, 01, 02, 03 . . . 23
%I	Hour as number (12-hour).	01, 02, 03 . . . 12
%j	Day of year.	001, 002, 003 . . . 366
%m	Month as number.	01, 02, 03 . . . 12
%M	Minute as number.	00, 02, 03 . . . 59
%n	Newline character.	\n
%o	Time zone offset.	-06:00, -05:00, -04:00
%O	Time zone offset, corrected for daylight saving time.	-06:00, +05:00
%p	p.m. or a.m. on a 12-hour clock.	pm, am
%P	AM or PM on a 12-hour clock.	PM, AM
%r	Time for 12-hour clock.	12:00 PM (shortcut for %I:%M:%S %P)
%R	Time for 24-hour clock.	23:30 (shortcut for %H:%M)
%s	Seconds, including tenths.	30.10, 59.99
%S	Seconds as a number.	00, 01, 02, 03 . . . 59
%t	Tab character.	\t
%T	Time.	12:00:01, 21:32:00 (same as %H:%M:%S)
%w	Weekday as a number.	0, 1, 2, 3 . . . 6
%U	Number of the week in the year.	0, 1, 2, 3 . . . 51
%y	Year.	00, 01, 02 . . . 99
%Y	Year, including century.	1999, 2000, 2001
%%	Literal.	%

getDate()

Returns the date in one of a couple predetermined formats. When passed the `DATE_FORMAT_ISO` constant, the date returns in a string that looks like `2006-04-10 13:00:00`.

```
string getDate([integer $format = DATE_FORMAT_ISO])
```

Parameter	Type	Description
$format	integer	A value that specifies the format in which to return the date.

Date Formats

Constant	Date Format	Example
`DATE_FORMAT_ISO`	YYYY-MM-DD HH:MM:SS	`1997-08-29 02:14:00`
`DATE_FORMAT_ISO_BASIC`	YYYYMMDD➥ THHMMSS (Z|(+/-)HHMM)?	`19970829T02:10:00Z-0400`
`DATE_FORMAT_ISO_EXTENDED`	YYYY-MM-DDTHH:MM: SS(Z|(+/-)HH:MM)?	`1997-08-29T02:14:00Z-04:00`
`DATE_FORMAT_ISO_EXTENDED_MICROTIME`	YYYY-MM-SSTHH:MM: SS(.S*)?(Z|(+/-)HH:MM)?	`1997-08-29T02:14:00.0000000Z-04:00`
`DATE_FORMAT_TIMESTAMP`	YYYYMMDDHHMMSS	`19970829021400`
`DATE_FORMAT_UNIXTIME`	long integer	`872820840`

getDay()

Returns an integer that represents the number of the day in the month.

```
integer getDay()
```

getDayName()

Returns the name of the day in the week, such as Monday, Tuesday, Wednesday.

```
string getDayName([boolean $abbr = false] [, mixed $length = 3])
```

Parameter	Type	Description
$abbr	boolean	`true` if the day's name should be shortened.
$length	mixed	If specified, determines the maximum length of the day name returned.

getDayOfWeek()

Returns the number of the day in the week, starting with `0` on Sunday.

```
integer getDayOfWeek()
```

getDaysInMonth()

Returns the number of total days in the current month.

```
integer getDaysInMonth()
```

getHour()

Returns the `Date`'s hour as an integer from `00` to `23`.

```
integer getHour()
```

getJulianDate()

Returns the Julian date as an integer, which is the number of days since Monday, January 1, 4713 BC. You can find more information about Julian dates at `http://en.wikipedia.org/wiki/Julian_date`.

```
integer getJulianDate()
```

getMinute()

Returns the minute as an integer between `00` and `59`.

```
integer getMinute()
```

getMonth()

Returns the month as an integer between `01` and `12`.

```
integer getMonth()
```

getMonthName()

Returns the name of the month in the `Date`.

```
string getMonthName([boolean $abbr = false])
```

Parameter	Type	Description
$abbr	boolean	If `true`, the name of the month is abbreviated. An example is `Jan` instead of `January`.

getNextDay()

Returns a `Date` object that represents the day immediately following this one chronologically. The time portion of the new `Date` object is the same as the original object's time.

```
Date getNextDay()
```

getNextWeekday()

Returns a `Date` object that represents the weekday that follows this one chronologically. If the current `Date` object is a Friday, this method will return the following Monday represented as a `Date` object. The time portion of the new `Date` object is the same as the original object's time.

```
Date getNextWeekday()
```

getPrevDay()

Returns a `Date` object that represents the date immediately preceding this one chronologically. The time portion of the new `Date` object is the same as the original object's time.

```
Date getPrevDay()
```

getPrevWeekday()

Returns a `Date` object that represents the weekday (Monday through Friday) that immediately precedes this one. The time portion of the new `Date` object is the same as the original object's time.

```
Date getPrevWeekday()
```

getQuarterOfYear

Returns the `Date`'s quarter of the year as an integer, `1` through `4`.

```
integer getQuarterOfYear()
```

getSecond()

Returns the second in the `Date`'s time as an integer between `00` and `59`.

```
integer getSecond()
```

getTime()

Returns the time as an integer expressing the number of seconds since January 1, 1970.

```
integer getTime()
```

getWeekOfYear()

Returns the number of the current week in the year. The numbering starts at `1` with the first Sunday in the year.

```
integer getWeekOfYear()
```

getWeeksInMonth()

Returns the number of weeks in the current month.

```
integer getWeeksInMonth()
```

getYear()

Returns the current year as an integer. For April 1, 2007, the returned value is `2007`.

```
integer getYear()
```

inDaylightTime()

Returns `true` if the `Date` is in daylight saving time, taking time zone into account.

```
boolean inDaylightTime()
```

Caution See the sections "convertTZ()" and "convertTZbyID()" for limitations on time zones with different operating systems. Because this method takes time zone into account, it's subject to the same limitations.

isFuture()

Returns `true` if the `Date` represents a date and time that are in the future.

```
boolean isFuture()
```

isLeapYear()

Returns `true` if the `Date` represents a leap year.

```
boolean isLeapYear()
```

isPast()

Returns `true` if the `Date` represents a date and time that are in the past.

```
boolean isPast()
```

setDate()

Sets the value of the `Date` given the string, and optionally the format of the string.

```
void setDate(string $date [, integer $format = DATE_FORMAT_ISO])
```

Parameter	Type	Description
$date	string	A string representation of a date, such as `2005-04-11 12:00:00`.
$format	integer	A constant that describes the format of the given string. See the section “getDate()” for a list of these constants.

setDay()

Sets the day of the `Date`, expecting an integer `0` through `31`. If you specify a number outside this range, the value is set to `1`.

```
void setDay(integer $d)
```

Parameter	Type	Description
$d	integer	The number of the day in the month.

setHour()

Sets the hour of the `Date`, expecting an integer value `00` through `23`. If you specify a number out of that range, the hour is set to `00`.

```
void setHour(integer $h)
```

Parameter	Type	Description
$h	integer	The number of the hour in a 24-hour day.

setMinute()

Sets the minutes portion of the `Date`, expecting an integer with the value `0` through `59`. If you specify out of that range, the minute is set to `00`.

```
void setMinute(integer $m)
```

Parameter	Type	Description
$m	integer	The minute in the hour.

setMonth()

Sets the month of the `Date`, expressed as an integer between `0` and `12`. If you specify a number out of this range, the value is set to `1`.

```
void setMonth(integer $m)
```

Parameter	Type	Description
$m	integer	The month of the `Date` object.

setSecond()

Sets the second portion of the `Date`, expressed as an integer `0` through `59`. If you specify a number that's out of this range, the value is set to `0`.

```
void setSecond(integer $s)
```

Parameter	Type	Description
$s	integer	The second in the minute.

setTZ()

Sets the time zone of the `Date`. The date and time are both left unmodified. See the section "convertTZ()" to change the date and time to a different value. This method will call `setTZbyID()` if the parameter specified isn't a `Date_TimeZone` object.

```
void setTZ(Date_TimeZone $tz)
```

Parameter	Type	Description
$tz	Date_TimeZone	A `Date_TimeZone` object that contains all the time zone information.

setTZbyID()

Sets the time zone using a string identifier that indicates which time zone to set. Values might be CST, PST, EST, or EDT. If you specify a value that isn't valid, the system's default time zone is used.

```
void setTZbyID(string $id)
```

Parameter	Type	Description
$id	string	An identifier that indicates which time zone to set on the Date.

setYear()

Sets the year part of the Date.

```
void setYear(integer $y)
```

Parameter	Type	Description
$y	integer	The year to set on the Date.

subtractSeconds()

Subtracts the given number of seconds from the Date, modifying it and setting the Date's value to the result.

```
void subtractSeconds(integer $sec)
```

Parameter	Type	Description
$sec	integer	The number of seconds to subtract from the Date.

subtractSpan()

Subtracts the given Date_Span from the Date and sets the Date to the new value. Date_Span objects can be easier to use than subtracting seconds when subtracting large amounts of time, such as days, weeks, and so on.

```
void subtractSpan(Date_Span $span)
```

Parameter	Type	Description
$span	Date_Span	The Date_Span object to subtract from the Date.

toUTC()

Sets the Date to UTC time and sets the time zone to "UTC."

```
void toUTC()
```

toUTCbyOffset()

Sets the UTC time by using an offset, such as `-06:00`, `+04:00`, `+0800`, and `-1200`.

```
void toUTCbyOffset(mixed $offset)
```

Parameter	Type	Description
$offset	mixed	A string that indicates the time to offset first, then set the `Date`'s time zone to UTC.

Date_Span API

Date_Span Constructor

Creates a new instance of a `Date_Span` object, and initializes the value to the time given as an argument.

```
Date_Span(mixed $time [, mixed $format = null])
```

Parameter	Type	Description
$time	mixed	A value representing a time. Most commonly, this is a string formatted in either the default input format or the one specified by `$format`.
$format	mixed	The format used by `$time`.

add()

Adds the provided `Date_Span` object to the current `Date_Span`.

```
void add(Date_Span $time)
```

Parameter	Type	Description
$time	Date_Span	The `Date_Span` to add.

compare()

Compares two `Date_Span` objects. If `$time1` is longer than `$time2`, the result is `1`. If `$time2` is longer than `$time1`, the result is `-1`. The result is `0` if they're equal, which makes this method ideal for sorting algorithms.

```
integer compare(Date_Span $time1, Date_Span $time2)
```

Parameter	Type	Description
$time1	Date_Span	The first `Date_Span` to compare.
$time2	Date_Span	The second `Date_Span` to compare.

copy()

Copies the value of the `Date_Span` into the given `Date_Span` object and returns `true` if the operation was successful.

```
boolean copy(Date_Span $time)
```

Parameter	Type	Description
$time	Date_Span	The `Date_Span` that the current `Date_Span` will be copied into.

equal()

Returns `true` if the `Date_Span` is equal to the provided `Date_Span`.

```
boolean equal(Date_Span $time)
```

Parameter	Type	Description
$time	Date_Span	The `Date_Span` to compare.

format()

Formats the `Date_Span` as a string.

```
string format([string $format = null])
```

Parameter	Type	Description
$format	string	The format to use when representing the `Date_Span` as a string.

Format strings for Date_Span

Format	Description	Example
`%C`	Days with time, same as `%D, %H:%M:%S`.	`2, 02:15:30`
`%d`	Total days as a float number.	`1.25` (1 day, 6 hours)
`%D`	Days as a decimal number.	`2` (2 days)
`%e`	Total hours as a float number.	`23.25`
`%f`	Total minutes as a float number.	`2.5` (2 minutes, 30 seconds)
`%g`	Total seconds as a decimal number.	`75` (1 minute, 15 seconds)
`%h`	Hours as a decimal number.	`3`
`%H`	Hours as a decimal number limited to two digits.	`03s`
`%m`	Minutes as a decimal number.	`5`
`%M`	Minutes as a decimal number limited to two digits.	`05`
`%n`	Newline character (`\n`).	`\n`
`%p`	Either "am" or "pm"—case insensitive—depending on the time. If "pm" is detected, it adds 12 hours to the resulting time span without any checks.	`pm`
`%r`	Time in am/pm notation, same as `%H:%M:%S %p`.	`02:15:30 pm`
`%R`	Time in 24-hour notation, same as `%H:%M`.	`14:15`
`%s`	Seconds as a decimal number.	`50`
`%S`	Seconds as a decimal number limited to two digits.	`03`
`%t`	Tab character (`\t`).	`\t`
`%T`	Current time equivalent, same as `%H:%M:%S`.	`04:22:00`
`%%`	Literal "%."	`%`

getDefaultFormat()

Returns the default format of the `Date_Span`.

```
mixed getDefaultFormat()
```

getDefaultInputFormat()

Returns the default input format of the `Date_Span`.

```
mixed getDefaultInputFormat()
```

greater()

Returns `true` if this `Date_Span` is greater than the provided `Date_Span`.

```
boolean greater(Date_Span $time)
```

Parameter	Type	Description
$time	Date_Span	The `Date_Span` to compare.

greaterEqual()

Returns `true` if this `Date_Span` is either greater than or equal to the provided `Date_Span`.

```
boolean greaterEqual(object Date_Span $time)
```

Parameter	Type	Description
$time	Date_Span	The `Date_Span` to compare.

isEmpty()

Returns `true` if the `Date_Span` is empty.

```
boolean isEmpty()
```

lower()

Returns `true` if this `Date_Span` represents a shorter amount of time than the provided `Date_Span`.

```
boolean lower(object Date_Span $time)
```

Parameter	Type	Description
$time	Date_Span	The `Date_Span` to compare.

lowerEqual()

Returns `true` if this `Date_Span` contains a shorter or equal time span than the `Date_Span` provided.

```
boolean lowerEqual(object Date_Span $time)
```

Parameter	Type	Description
$time	Date_Span	The `Date_Span` to compare.

set()

Sets the `Date_Span` using the given time and optional format.

```
boolean set(mixed $time [, mixed $format = null])
```

Parameter	Type	Description
$time	mixed	A value representing the time for the `Date_Span`.
$format	mixed	The format to use when parsing `$time` into the `Date_Span` value. See the `format()` method for available formats. Optional.

setDefaultFormat()

Sets the default format to use when returning the value of the `Date_Span` as a formatted string. It returns the last used format.

```
mixed setDefaultFormat(mixed $format)
```

Parameter	Type	Description
$format	mixed	The format to use as the new default. The default is `%C`. For more information about the formats, see the `format()` method.

setDefaultInputFormat()

Sets the default input format to use when providing strings to set the value of the `Date_Span`.

```
mixed setDefaultInputFormat(mixed $format)
```

Parameter	Type	Description
$format	mixed	The default input format to use.

setFromArray()

Sets the value of the `Date_Span` using the items in the provided array. The first element in the array holds seconds, the second minutes, the third hours, and the fourth days. Zeros are used for any elements that are missing.

```
boolean setFromArray(array $time)
```

Parameter	Type	Description
$time	array	The new `Date_Span` value to set as an array.

setFromDateDiff()

Sets the value of the Date_Span to be equal to the difference between the two Date objects provided. true is returned if the method is successful.

```
boolean setFromDateDiff(object Date $date1, object Date $date2)
```

Parameter	Type	Description
$date1	Date	The first Date to use in the difference.
$date2	Date	The Date to subtract from the first date to get a time span value to set the Date_Span.

setFromDays()

Sets the value of the Date_Span to the number of days provided. Returns true on success.

```
boolean setFromDays(float $days)
```

Parameter	Type	Description
$days	float	The number of days to use for the value of the Date_Span.

setFromHours()

Sets the value of the Date_Span to the number of hours provided. Returns true on success.

```
boolean setFromHours(float $hours)
```

Parameter	Type	Description
$hours	float	The number of hours to use for value of the Date_Span.

setFromMinutes()

Sets the value of the Date_Span to the number of minutes provided. Returns true on success.

```
boolean setFromMinutes(float $minutes)
```

Parameter	Type	Description
$minutes	float	The number of hours to use for the value of the Date_Span.

setFromSeconds()

Sets the value of the `Date_Span` to the number of seconds provided. Returns `true` on success.

```
boolean setFromSeconds(integer $seconds)
```

Parameter	Type	Description
$seconds	integer	The number of seconds to use for the value of the `Date_Span`.

setFromString()

Sets the value of the `Date_Span` given a formatted string. The format is optional, and if it's not provided, the `Date_Span` will use the default input string format.

```
boolean setFromString(string $time [, string $format = null])
```

Parameter	Type	Description
$time	string	The time duration for the `Date_Span` expressed as a string.
$format	string	A string that describes the format of the `$time` string.

■ **Note** The `setFromString()` method always uses the last provided values. This is important when providing times that might override each other, such as `setFromString('06, 1', '%M %m')`. In this case, the time span is one minute. The method doesn't add time to the existing values.

subtract()

Subtracts the value of the `Date_Span` provided from the current `Date_Span`. If the time span to subtract is larger than the original, the result is zero.

```
void subtract(Date_Span $time)
```

Parameter	Type	Description
$time	Date_Span	The value to subtract from the current `Date_Span`.

toDays()

Returns the `Date_Span` as a value in days.

```
float toDays()
```

toHours()

Returns the `Date_Span` as a value in hours.

```
float toHours()
```

toMinutes()

Returns the `Date_Span` as a value in minutes.

```
float toMinutes()
```

toSeconds()

Returns the `Date_Span` as a value in seconds.

```
integer toSeconds()
```

Date_TimeZone API

Date_TimeZone Constructor

Creates a new `Date_TimeZone` object using the given identifier. The identifier is a name that uniquely identifies the time zone. You can obtain the list of IDs using the `getAvailableIDs()` method.

```
Date_TimeZone Date_TimeZone(string $id)
```

Parameter	Type	Description
$id	string	An identifier for the time zone. See the `getAvailableIDs()` method to get a list of the available IDs.

getAvailableIDs()

Returns a list of the available identifier strings that you can use to create new `Date_TimeZone` objects.

```
mixed getAvailableIDs()
```

getDefault()

Returns a `Date_TimeZone` with the same time zone as the system's default time zone.

```
Date_TimeZone getDefault()
```

getDSTLongName()

Returns the long name of the of the time zone, including the daylight saving time information, such as Mountain Daylight Time.

```
string getDSTLongName()
```

getDSTSavings()

Returns the number of milliseconds of time offset for daylight saving time. It's always 3600000.

```
integer getDSTSavings()
```

getDSTShortName()

Returns the short name of the time zone, including the daylight saving time information; for example, PDT for Pacific Daylight Time.

```
string getDSTShortName()
```

getID()

Returns the string identifier of the `Date_TimeZone` object.

```
string getID()
```

getLongName()

Returns the long name of the `Date_TimeZone` object.

```
string getLongName()
```

getOffset()

Returns the offset from UTC for the time zone, taking into account the offset for daylight saving time, if applicable.

```
integer getOffset(Date $date)
```

Parameter	Type	Description
$date	Date	The `Date` from which to get the offset.

getRawOffset()

Returns the offset from UTC for the time zone, without taking into account daylight saving time.

```
integer getRawOffset()
```

getShortName()

Returns the short name of the `Date_TimeZone` object. Example: `CST`, `DST`, `GMT-12:00`, `WST`.

```
string getShortName()
```

hasDaylightTime()

Returns `true` if the time zone observes daylight saving time.

```
boolean hasDaylightTime()
```

inDaylightTime()

Returns `true` if the date is in daylight saving time. This method isn't necessarily reliable on Windows systems because it uses OS calls to get the result.

```
boolean inDaylightTime(Date $date)
```

Parameter	Type	Description
$date	Date	The `Date` to test to see if it's observing daylight saving time.

isEqual()

Returns `true` if the `Date_TimeZone` object is equal to the provided `Date_TimeZone`.

```
boolean isEqual(Date_TimeZone $tz)
```

Parameter	Type	Description
$tz	Date_TimeZone	The `Date_TimeZone` to test.

isEquivalent()

Returns `true` if the `Date_TimeZone` has an offset that's equal and offset of the provided `Date_TimeZone`. For them to be equivalent, both must have the same observation of daylight saving time (either they both do, or both don't observe it).

```
boolean isEquivalent(Date_TimeZone $tz)
```

Parameter	Type	Description
$tz	Date_TimeZone	The `Date_TimeZone` to test.

isValidID()

Returns `true` if the provided string is a valid identifier for one of the time zones stored in the `Date_TimeZone` data.

```
boolean isValidID(string $id)
```

Parameter	Type	Description
$id	string	The identifier to look for in the list of available identifiers.

setDefault()

Sets the default time zone.

```
void setDefault(string $id)
```

Parameter	Type	Description
$id	string	The identifier of the time zone to set as default.

Examples

Converting UTC to a Different Time Zone

This example shows how to create a new `Date` object given a formatted string that represents a date and time, then use a `Date_TimeZone` object to convert the time zone on the `Date`. The `format()` method shown at the end of the example demonstrates how to show the `Date` in a somewhat user-friendly output string.

```
<?php
/* Converting a UTC date to a time zone */

require_once 'Date.php';

$date = new Date('2006-04-10 13:00:00');
$date->toUTC();
/* This will print the date out, which is now a UTC date */
echo $date->getDate(DATE_FORMAT_ISO) . "\n";

$cst = new Date_TimeZone('CST');

/* Now convert the date to the new time zone. */
$date->convertTZ($cst);
/* This will print that date again, this time in CST (UTC -6) */
echo $date->getDate(DATE_FORMAT_ISO) . "\n";
```

```
/* This will print the same date, but in a more friendly
 * format. The date will look like: Monday, April 4 2006 at 07:00 am.
 */
echo $date->format("%A, %B %e %Y at %I:%M %p%n");

?>
```

When you run the preceding example, the output will look like this:

```
2006-04-10 13:00:00
2006-04-10 07:00:00
Monday, April 10 2006 at 07:00 am
```

Adding Dates and Times

This example demonstrates how to do a little bit more than simply add a few days to a given date. In this example, a `Date_Span` object is used to add a certain number of days, but then the `getNextWeekday()` method is called to get the weekday that falls on the next weekday (Monday through Friday) after the dates have been added. Why go to all this trouble? It's handy if you're calculating times, such as for payroll processing, which might be done on the third business day after month closing.

```
<?php
/* Adding Dates and Times. */

require_once 'Date.php';
require_once 'Date/Span.php';

$date = new Date('2006-04-05');
$date->setTZbyID('CST');
echo $date->format("Original date:  %A, %B %e %Y %n");

/* Now add three days to the current day */
$span = new Date_Span();
$span->setFromDays(3);

$date->addSpan($span);
echo $date->format("Three days later:  %A, %B %e %Y %n");

/* Now find the next business day */
$nextBusinessDate = $date->getNextWeekday();
echo $nextBusinessDate->format("The next weekday:  %A, %B %e %Y %n");
?>
```

You'll see the following output when you run the preceding example:

```
Original date:  Wednesday, April 5 2006
Three days later:  Saturday, April 8 2006
The next weekday:  Monday, April 10 2006
```

Comparing Dates

This example contains a potpourri of different date comparison methods. First, the `isPast()` method is used to compare the date to today's date. Then, the `Date::compare()` method is used to show the integer value of comparing two dates.

```
<?php

/* Comparing dates */

require_once 'Date.php';

$date = new Date('2006-02-10');
echo $date->format("Date is:  %D%n");
echo sprintf("The date is %s today.\n",
    $date->isPast() ? "before" : "after");

/* Set the year to something in the future */
$date->setYear(2015);
echo $date->format("Date is:  %D%n");
echo sprintf("The date is %s today.\n",
    $date->isPast() ? "before" : "after");

$now = new Date();

/* Compare the future date with now. */
$result = Date::compare($now, $date);
echo sprintf('Comparing $now with $date:  %s', $result) . "\n";

$result = Date::compare($date, $now);
echo sprintf('Comparing $date with $now:  %s', $result) . "\n";
?>
```

The output of running the preceding example looks like this:

```
Date is:  02/10/2006
The date is before today.
Date is:  02/10/2015
The date is after today.
Comparing $now with $date:  -1
Comparing $date with $now:  1
```

Date_Holidays

The Date_Holidays package contains classes that allow you to get information about holidays during the course of a year. The `Date_Holidays` class includes a `factory()` method that returns a `Date_Holidays_Driver`. This driver class contains methods you can use to see if a particular date is a holiday, grab the holidays that fall within date ranges, and more.

Common Uses

Following are common uses of the Date_Holidays package:

- Printing out the dates of holidays throughout the year
- Checking to see if a given date falls on a holiday

Related Packages

The following related package is found in this book:

Date

Dependencies

Following are the dependencies for the Date_Holidays package.

Required Packages

XML_Serializer

Optional Packages

Console_Getargs

Date_Holidays API

Date_Holidays Constructor

Creates a new instance of a `Date_Holidays` object. Instead of using the constructor to create a new instance of the object, use the `factory()` method.

```
Date_Holidays Date_Holidays()
```

errorsOccurred()

Returns `true` if errors have occurred while attempting to create the `Date_Holidays_Driver` class.

```
boolean errorsOccurred()
```

factory()

Builds and returns an instance of an object that inherits the `Date_Holidays_Driver` class with the specified driver ID.

```
Date_Holidays_Driver factory(string $driverId [, string $year = null]
                              [, string $locale = null] [, mixed $external = false])
```

Parameter	Type	Description
$driverId	string	A string that identifies the driver to load. See the `getInstalledDrivers()` method to get a list of the drivers you can use.
$year	string	The year you'd like to use for the holidays.
$locale	string	The locale to use, such as `en_US`.
$external	mixed	`true` if the specified driver ID is for a driver that isn't loaded with the base installation of the package.

factoryISO3166()

Builds and returns an instance of an object that inherits from the `Date_Holiday_Driver` class that conforms to the given International Standards Organization (ISO) code. The ISO codes are documented at `http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html`.

```
Date_Holidays factoryISO3166(string $isoCode [, string $year = null]
                              [, string $locale = null] [, mixed $external = false])
```

Parameter	Type	Description
$isoCode	string	The ISO 3166 code to use for loading the driver. Examples are FR, GB, and US for France, the United Kingdom, and the United States, respectively.
$year	string	The year for the holidays that are found by the driver.
$locale	string	The language locale to use, such as en_US.
$external	mixed	true if the specified locale is not in the standard package.

getErrors()

Returns an array of errors.

```
array getErrors([boolean $purge = false])
```

Parameter	Type	Description
$purge	boolean	If true, the existing errors are purged after they're returned.

getErrorStack()

Returns the error stack.

```
PEAR_ErrorStack &getErrorStack()
```

getInstalledDrivers()

Returns an array of the available driver names. The `getInstalledDrivers()` method does this by looking into its `Driver` directory and returning the files found, minus the `.php` extension.

```
array getInstalledDrivers([string $directory = null])
```

Parameter	Type	Description
$directory	string	The directory to look in for the drivers.

getInstalledFilters()

Similar to the `getInstalledDrivers()` method, this one parses through the `Filters` directory and returns the filters contained in it. Both these methods are handy if you don't have access to the file system, and therefore can't determine the names of the drivers and files.

```
array getInstalledFilters([string $directory = null])
```

Parameter	Type	Description
$directory	string	The directory to look in for the filters.

isError()

Returns `true` if the object passed in is an error object.

```
boolean isError(mixed $data [, integer $code = null])
```

Parameter	Type	Description
$data	mixed	The object to inspect to see if it is an error.
$code	integer	The error code to look for in the object.

raiseError()

Raises an error with the specified code and optional message.

```
PEAR_Error raiseError(integer $code [, string $msg = null])
```

Parameter	Type	Description
$code	integer	The code to use for the error.
$msg	string	The message that will be included in the new error.

staticGetProperty()

Returns the value of the given property without having an instance created of the `Date_Holidays` class.

```
mixed staticGetProperty(string $prop)
```

Parameter	Type	Description
$prop	string	The name of the property to return.

staticSetProperty()

Assigns the value of the given property without having an instance of the `Date_Holidays` class created.

```
void staticSetProperty(string $prop, string $value)
```

Parameter	Type	Description
$prop	string	The name of the property to set.
$value	string	The value that the property should be set to.

Date_Holidays_Driver API

Date_Holidays_Driver Constructor

Don't use the constructor to create an instance of the driver—instead use the `factory()` method on the `Date_Holidays` class.

```
Date_Holidays_Driver Date_Holidays_Driver()
```

addCompiledTranslationFile()

Makes the contents of the given file available to the driver.

```
boolean addCompiledTranslationFile(string $file, string $locale)
```

Parameter	Type	Description
$file	string	The name of the translation file.
$locale	string	The name of the locale for the translation.

addDriver()

Adds a driver.

```
void addDriver(object Date_Holidays_Driver $driver)
```

Parameter	Type	Description
$driver	Date_Holidays_Driver	A driver object that's added to load holidays.

Note The Date_Holidays package is in alpha status, so some of the methods are only partially implemented. As of the time of this writing, the `addDriver()` method is blank.

addTranslationFile()

Adds a file's content. The information in the file is available for the specified locale.

```
boolean addTranslationFile(string $file, string $locale)
```

Parameter	Type	Description
$file	string	The name of the file to load.
$locale	string	The locale contained in the file.

getHoliday()

Returns a `Date_Holidays_Holiday` object that's identified by the internal name.

```
Date_Holidays_Holiday getHoliday(string $internalName [, string $locale = null])
```

Parameter	Type	Description
$internalName	string	The internal designation for a holiday.
$locale	string	The locale to use when loading holiday information.

getHolidayDate()

Returns a `Date` that represents the given holiday name. The `Date` includes the month, day, and year of the holiday. If an error occurs, the method returns `PEAR_Error`.

```
Date getHolidayDate(string $internalName)
```

Parameter	Type	Description
$internalName	string	An internal designation for the holiday.

getHolidayDates()

Returns an array of `Date` objects using the specified filter.

```
array getHolidayDates([Date_Holidays_Filter $filter = null])
```

Parameter	Type	Description
$filter	Date_Holidays_Filter	A `Date_Holidays_Filter` object to use when retrieving the holidays.

getHolidayForDate()

Returns a `Date_Holidays_Holiday` object for the given date.

```
object getHolidayForDate(mixed $date [, string $locale = null]
                              [, boolean $multiple = false])
```

Parameter	Type	Description
$date	mixed	A string or object that contains a date.
$locale	string	The locale to use when loading the holiday information.
$multiple	boolean	If `true`, the method will load more than one holiday.

getHolidayProperties()

Returns an array of properties for the holiday identified by `$internalName`, or an empty array if properties are found.

```
array getHolidayProperties(string $internalName [, string $locale = null])
```

Parameter	Type	Description
$internalName	string	The internal name for the holiday.
$locale	string	The locale to use when loading information about the holiday.

getHolidays()

Returns an array of `Date_Holidays_Holiday` objects that match the specified filter, or a `PEAR_Error` object if an error occurred.

```
array getHolidays([Date_Holidays_Filter $filter = null] [, string $locale = null])
```

Parameter	Type	Description
$filter	Date_Holidays_Filter	The filter to use when retrieving holiday information.
$locale	string	The name of the locale to use when loading information about the holidays.

getHolidaysForDatespan()

Returns an array of `Date_Holidays_Holiday` objects for the holidays that occur between the two specified dates. If no holidays are found, then an empty array will be returned.

```
array getHolidaysForDatespan(mixed $start, mixed $end
                              [, Date_Holidays_Filter $filter = null]
                              [, string $locale = null])
```

Parameter	Type	Description
$start	mixed	The starting date of the time span.
$end	mixed	The end date of the time span.
$filter	Date_Holidays_Filter	The filter to use for loading holidays.
$locale	string	The locale to use when loading holiday information.

getHolidayTitle()

Returns the title of the holiday.

```
string getHolidayTitle(string $internalName [, string $locale = null])
```

Parameter	Type	Description
$internalName	string	An internal designation for the holiday.
$locale	string	The name of the locale to use for holiday information.

getHolidayTitles()

Returns an array of holiday titles that match the specified filter.

```
array getHolidayTitles([Date_Holidays_Filter $filter = null]
                              [, string $locale = null])
```

Parameter	Type	Description
$filter	Date_Holidays_Filter	The filter to use when retrieving holiday titles.
$locale	string	The locale to use for holiday information.

getInternalHolidayNames()

Returns an array of internal holiday names for the driver that's loaded.

```
array getInternalHolidayNames()
```

getISO3166Codes()

Returns an array of ISO 3166 codes for the holidays known by the driver that's loaded.

```
array getISO3166Codes()
```

Note The Date_Holidays package is in alpha status, so some of the methods are only partially implemented. As of the time of this writing, the `getISO3166Codes()` method simply returns an empty array.

getYear()

Returns the year for the current driver. When the factory loads the driver, you can specify a year. The driver loads the holidays that occur during that year.

```
integer getYear()
```

isHoliday()

Returns `true` if the given date is a holiday, or `PEAR_Error` if an error occurs.

```
boolean isHoliday(mixed $date [, Date_Holidays_Filter $filter = null])
```

Parameter	Type	Description
$date	mixed	The date that's being checked for holidays.
$filter	Date_Holidays_Filter	The filter to apply when checking the date.

removeDriver()

Removes the specified driver from the list of loaded drivers.

```
boolean removeDriver(Date_Holidays_Driver $driver)
```

Parameter	Type	Description
$driver	Date_Holidays_Driver	The driver to remove from the list of holiday drivers.

Note As of the time of this writing, the `removeDriver()` method is blank.

setLocale()

Allows you to set the locale for which the driver will find and name holidays.

```
void setLocale(string $locale)
```

Parameter	Type	Description
$locale	string	The name of the locale to use for holiday information.

setYear()

Sets the year for holidays that are found by the driver, rebuilding the holidays for the given year.

```
boolean setYear(integer $year)
```

Parameter	Type	Description
$year	integer	The year during which the holidays will occur. If 2003 is specified, the methods for the driver will return holidays that are specific to the year 2003.

Examples

Determining the Date of a US Holiday

This example demonstrates how to determine the dates for two US holidays in a particular year. The first holiday is formatted to display the long month name and the date. The second holiday is formatted to display the weekday on which the holiday falls.

```
<?php
/* Determining the date of a US holiday */

require_once 'Date/Holidays.php';

$holidays = &Date_Holidays::factory('USA', 2007, 'en_EN');
if (Date_Holidays::isError($holidays)) {
    die('Factory was unable to produce driver-object');
}

$mlkDay = &$holidays->getHoliday('mlkDay', 'en_EN');
$date = $mlkDay->getDate();

echo $date->format("In %Y, Martin Luther King Day is on %B %d.%n");
```

```
$independenceDay = &$holidays->getHoliday('independenceDay', 'en_EN');
$idDate = $independenceDay->getDate();
echo $idDate->format("In %Y, Independance Day is on a %A.%n");

?>
```

The output of this example is as follows:

```
In 2007, Martin Luther King Day is on January 15.
In 2007, Independance Day is on a Wednesday.
```

Determining If a Day Is a Holiday

Using a specific date, this example checks to see if the date is a holiday.

```
<?php
/* Determining if a day is a holiday */

require_once 'Date/Holidays.php';

$holidays = &Date_Holidays::factory('Christian', 2008, 'en_EN');
if (Date_Holidays::isError($holidays)) {
    die('Factory was unable to produce driver-object');
}

$date = new Date('2008-12-25');

if ($holidays->isHoliday($date)) {
    echo $date->format('%D is a holiday!%n');
} else {
    echo $date->format('%D is NOT a holiday!%n');
}

?>
```

The output is as follows:

```
12/25/2008 is a holiday!
```

Getting the Holidays Between Two Dates

This example uses a range of dates to find out which holidays are within the date range. The `getTitle()` method displays the title of the holiday, and the `format()` method displays the day the holiday occurs.

```php
<?php

/* Getting the holidays between two dates. */

require_once 'Date/Holidays.php';

$holidays = &Date_Holidays::factory('USA');
if (Date_Holidays::isError($holidays)) {
    die('Factory was unable to produce driver-object');
}
$holidays->setLocale('en_EN');

$dates = $holidays->getHolidaysForDatespan('2006-04-01', '2006-08-01');
$limit = count($dates);

echo "Looking for holidays between '2006-04-01' and '2006-08-01':\n";

for ($i = 0; $i < $limit; $i++) {

    $date = $dates[$i]->getDate();

    printf("\t\"%s\" is on %s\n",
        $dates[$i]->getTitle(),
        $date->format("%B %d"));
}

?>
```

The output of this example is as follows:

```
Looking for holidays between '2006-04-01' and '2006-08-01':
    "Memorial Day" is on May 29
    "Independence Day" is on July 04
```

Numbers_Roman

The Numbers_Roman package, although smaller in API, is handy for making the translation between Roman numerals and decimal numbers.

Common Uses

The Numbers_Roman package is useful for the following purposes:

- Interesting display of copyright dates
- Using Roman numerals in titles and text
- Parsing Roman numerals into decimal numbers

Related Packages

The following package found in this book is related to Numbers_Roman:

Numbers_Words

Dependencies

Numbers_Roman depends on the following packages.

Required Packages

None

Optional Packages

None

API

toNumber()

Converts a given string, expressed as a Roman numeral such as `IV` or `XII`, into a decimal number such as `4` or `12`.

```
integer toNumber(string $roman)
```

Parameter	Type	Description
$roman	string	A Roman numeral as a string.

toNumeral()

Converts a decimal number into a string that represents a Roman numeral; for example, `6` is converted to `VI`.

```
string toNumeral(integer $num, [boolean $uppercase = true], [boolean $html = true])
```

Parameter	Type	Description
$num	integer	A decimal number, such as `10`, `100`, or `1211`.
$uppercase	boolean	If `true`, the Roman numeral string that's returned will be in uppercase.
$html	boolean	If `true`, the string that is returned will use HyperText Markup Language (HTML) formatting to place the bars above and below the numbers.

Examples

Converting to Roman Numerals

This example shows how you can convert decimal numbers into Roman numerals, in this case allowing you to be fancy with displaying a copyright notice by converting the year into a Roman numeral.

```
<?php
/* Converting to Roman Numbers */

require_once 'Numbers/Roman.php';
/* Getting fancy with turning years into Roman numerals. */
printf("Copyright (c) %s\n",
    Numbers_Roman::toNumeral(2006, true, false));

?>
```

Following is the output of the example:

```
Copyright (c) MMVI
```

Converting Roman Numerals to Decimal Numbers

One of the authors of this book is not known for being a sports fanatic. This example shows how to convert the Roman numerals, as used in the name of some famous sporting event in the US, into a decimal number.

```
<?php
/* Convert Roman Numerals to numbers */

require_once 'Numbers/Roman.php';

/* What was that Super Bowl, anyway? */

printf("Super Bowl XXXIV is number %s\n",
    Numbers_Roman::toNumber('XXXIV'));

?>
```

Here's the output of the preceding example:

```
Super Bowl XXXIV is number 34
```

Numbers_Words

The Numbers_Words package converts decimal numbers into words.

Common Uses

You can use Numbers_Words for the following purposes:

- Converting a number such as 100 into its text equivalent: one hundred
- Converting currency into text

Related Packages

The following package in this book is similar to Numbers_Words:

Numbers_Roman

Dependencies

Numbers_Words depends on the following other packages.

Required Packages

None

Optional Packages

None

API

raiseError()

Raises a PEAR error with the specified message. For performance reasons, the `PEAR.php` file is dynamically included in this method.

```
void raiseError(string $msg)
```

toCurrency()

Returns a string that represents the currency that's passed as an argument to the method, given the locale.

```
string toCurrency(float $num [, string $locale = 'en_US']
                              [, string $integer_curr = ''])
```

Parameter	Type	Description
$num	float	The currency expressed as a number, such as `1.50`, `1,200.50`, or `1,000`.
$locale	string	The locale to use when interpreting the currency.
$integer_curr	string	A three-digit international currency code to use, as defined by the ISO 4217 standard.

toWords()

Returns a string that represents the number in words.

```
string toWords(integer $num [, string $locale = 'en_US'])
```

Parameter	Type	Description
$num	integer	The number to convert to words.
$locale	string	The name of the locale to use when assembling the words.

Examples

Converting Numbers into Words

This example demonstrates how to turn numbers into words.

```
<?php
/* Converting numbers to words */
require_once 'Numbers/Words.php';

/* Print a bunch of different numbers out to see what they
 * say...
 */
printf("%s\n", Numbers_Words::toWords(1));
printf("%s\n", Numbers_Words::toWords(10));
printf("%s\n", Numbers_Words::toWords(100));
printf("%s\n", Numbers_Words::toWords(1000));
printf("%s\n", Numbers_Words::toWords(1002));
printf("%s\n", Numbers_Words::toWords(1357));

?>
```

The output of the example is as follows:

```
one
ten
one hundred
one thousand
one thousand two
one thousand three hundred fifty-seven
```

Converting Currency into Words

The example here shows how to turn numbers into words, using the `toCurrency()` method.

```
<?php
/* Converts currency to words */
require_once 'Numbers/Words.php';

printf("%s\n", Numbers_Words::toCurrency(1.50));
printf("%s\n", Numbers_Words::toCurrency(10.99));
printf("%s\n", Numbers_Words::toCurrency(100.20));
printf("%s\n", Numbers_Words::toCurrency(17999.99));

?>
```

Here's the output:

```
one dollar fifty cents
ten dollars ninety-nine cents
one hundred dollars twenty cents
seventeen thousand nine hundred ninety-nine dollars ninety-nine cents
```