Date Class

Contains methods for the Date primitive data type.

Namespace

System

Usage

For more information on Dates, see Date Data Type.

Date Methods

The following are methods for `Date`.

- **addDays(additionalDays)**
  Adds the specified number of additional days to a Date.
- **addMonths(additionalMonths)**
  Adds the specified number of additional months to a Date
- **addYears(additionalYears)**
  Adds the specified number of additional years to a Date
- **day()**
  Returns the day-of-month component of a Date.
- **dayOfYear()**
  Returns the day-of-year component of a Date.
- **daysBetween(secondDate)**
  Returns the number of days between the Date that called the method and the specified date.
- **daysInMonth(year, month)**
  Returns the number of days in the month for the specified *year* and *month* (1=Jan).
- **format()**
  Returns the Date as a string using the locale of the context user
- **isLeapYear(year)**
  Returns `true` if the specified year is a leap year.
- **isSameDay(dateToCompare)**
  Returns `true` if the Date that called the method is the same as the specified date.
- **month()**
  Returns the month component of a Date (1=Jan).
- **monthsBetween(secondDate)**
  Returns the number of months between the Date that called the method and the specified date, ignoring the difference in days.
- **newInstance(year, month, day)**
  Constructs a Date from Integer representations of the *year*, *month* (1=Jan), and *day*.
- **parse(stringDate)**
  Constructs a Date from a String. The format of the String depends on the local date format.

- **toStartOfWeek()**
  Returns the start of the week for the Date that called the method, depending on the context user's locale.
- **valueOf(stringDate)**
  Returns a Date that contains the value of the specified String.
- **valueOf(fieldValue)**
  Converts the specified object to a `Date`. Use this method to convert a history tracking field value or an object that represents a `Date` value.
- **year()**
  Returns the year component of a Date

addDays(additionalDays)

Adds the specified number of additional days to a Date.

Signature

```
public Date addDays(Integer additionalDays)
```

Parameters

additionalDays

Type: Integer

Return Value

Type: Date

Example

```
Date myDate = Date.newInstance(1960, 2, 17);
Date newDate = mydate.addDays(2);
```

addMonths(additionalMonths)

Adds the specified number of additional months to a Date

Signature

```
public Date addMonths(Integer additionalMonths)
```

Parameters

additionalMonths

Type: Integer

Return Value

Type: Date

Example

```
date myDate = date.newInstance(1990, 11, 21);
date newDate = myDate.addMonths(3);
date expectedDate = date.newInstance(1991, 2, 21);
system.assertEquals(expectedDate, newDate);
```

```
public Date addYears(Integer additionalYears)
```

Parameters

additionalYears

Type: Integer

Return Value

Type: Date

Example

```
date myDate = date.newInstance(1983, 7, 15);
date newDate = myDate.addYears(2);
date expectedDate = date.newInstance(1985, 7, 15);
system.assertEquals(expectedDate, newDate);
```

day()

Returns the day-of-month component of a Date.

Signature

```
public Integer day()
```

Return Value

Type: Integer

Example

```
date myDate = date.newInstance(1989, 4, 21);
Integer day = myDate.day();
system.assertEquals(21, day);
```

dayOfYear()

Returns the day-of-year component of a Date.

Signature

```
public Integer dayOfYear()
```

Return Value

Type: Integer

Example

```
date myDate = date.newInstance(1998, 10, 21);
Integer day = myDate.dayOfYear();
system.assertEquals(294, day);
```

daysBetween(secondDate)

Parameters

secondDate

Type: Date

Return Value

Type: Integer

Usage

If the Date that calls the method occurs after the *secondDate*, the return value is negative.

Example

```
Date startDate = Date.newInstance(2008, 1, 1);
Date dueDate = Date.newInstance(2008, 1, 30);
Integer numberDaysDue = startDate.daysBetween(dueDate);
```

daysInMonth(year, month)

Returns the number of days in the month for the specified *year* and *month* (1=Jan).

Signature

```
public static Integer daysInMonth(Integer year, Integer month)
```

Parameters

year

Type: Integer

month

Type: Integer

Return Value

Type: Integer

Example

The following example finds the number of days in the month of February in the year 1960.

```
Integer numberDays = date.daysInMonth(1960, 2);
```

format()

Returns the Date as a string using the locale of the context user

Signature

```
public String format()
```

Return Value

Type: String

```
date myDate = date.newInstance(2001, 3, 21);
String dayString = myDate.format();
system.assertEquals('3/21/2001', dayString);
```

isLeapYear(year)

Returns `true` if the specified year is a leap year.

Signature

```
public static Boolean isLeapYear(Integer year)
```

Parameters

year

Type: Integer

Return Value

Type: Boolean

Example

```
system.assert(Date.isLeapYear(2004));
```

isSameDay(dateToCompare)

Returns `true` if the Date that called the method is the same as the specified date.

Signature

```
public Boolean isSameDay(Date dateToCompare)
```

Parameters

dateToCompare

Type: Date

Return Value

Type: Boolean

Example

```
date myDate = date.today();
date dueDate = date.newInstance(2008, 1, 30);
boolean dueNow = myDate.isSameDay(dueDate);
```

month()

Returns the month component of a Date (1=Jan).

Signature

```
public Integer month()
```

Return Value

```
date myDate = date.newInstance(2004, 11, 21);
Integer month = myDate.month();
system.assertEquals(11, month);
```

monthsBetween(secondDate)

Returns the number of months between the Date that called the method and the specified date, ignoring the difference in days.

Signature

```
public Integer monthsBetween(Date secondDate)
```

Parameters

secondDate

Type: Date

Return Value

Type: Integer

Example

```
Date firstDate = Date.newInstance(2006, 12, 2);
Date secondDate = Date.newInstance(2012, 12, 8);
Integer monthsBetween = firstDate.monthsBetween(secondDate);
System.assertEquals(72, monthsBetween);
```

newInstance(year, month, day)

Constructs a Date from Integer representations of the *year*, *month* (1=Jan), and *day*.

Signature

```
public static Date newInstance(Integer year, Integer month, Integer day)
```

Parameters

year

Type: Integer

month

Type: Integer

day

Type: Integer

Return Value

Type: Date

Example

The following example creates the date February 17th, 1960:

```
Date myDate = date.newinstance(1960, 2, 17);
```

```
public static Date parse(String stringDate)
```

Parameters

stringDate

Type: String

Return Value

Type: Date

Example

The following example works in some locales.

```
date mydate = date.parse('12/27/2009');
```

today()

Returns the current date in the current user's time zone.

Signature

```
public static Date today()
```

Return Value

Type: Date

toStartOfMonth()

Returns the first of the month for the Date that called the method.

Signature

```
public Date toStartOfMonth()
```

Return Value

Type: Date

Example

```
date myDate = date.newInstance(1987, 12, 17);
date firstDate = myDate.toStartOfMonth();
date expectedDate = date.newInstance(1987, 12, 1);
system.assertEquals(expectedDate, firstDate);
```

toStartOfWeek()

Returns the start of the week for the Date that called the method, depending on the context user's locale.

Signature

```
public Date toStartOfWeek()
```

For example, the start of a week is Sunday in the United States locale, and Monday in European locales. For example:

```
Date myDate = Date.today();
Date weekStart = myDate.toStartofWeek();
```

valueOf(stringDate)

Returns a Date that contains the value of the specified String.

Signature

```
public static Date valueOf(String stringDate)
```

Parameters

stringDate

Type: String

Return Value

Type: Date

Usage

The specified string should use the standard date format “yyyy-MM-dd HH:mm:ss” in the local time zone.

Example

```
string year = '2008';
string month = '10';
string day = '5';
string hour = '12';
string minute = '20';
string second = '20';
string stringDate = year + '-' + month
 + '-' + day + ' ' + hour + ':' +
minute + ':' + second;

Date myDate = date.valueOf(stringDate);
```

valueOf(fieldValue)

Converts the specified object to a `Date`. Use this method to convert a history tracking field value or an object that represents a `Date` value.

Signature

```
public static Date valueOf(Object fieldValue)
```

Parameters

fieldValue

Type: Object

Return Value

Type: Date

Example

This example converts history tracking fields to `Date` values.

```
List<AccountHistory> ahlist = [SELECT Field,OldValue,NewValue FROM AccountHistory];
for(AccountHistory ah : ahlist) {
  System.debug('Field: ' + ah.Field);
  if (ah.field == 'MyDate__c') {
    Date oldValue = Date.valueOf(ah.OldValue);
    Date newValue = Date.valueOf(ah.NewValue);
  }
}
```

Versioned Behavior Changes

`Date.valueOf` has been versioned in these releases.

API version 33.0 or earlier

If you call `Date.valueOf` with a `Datetime` object, the method returns a `Date` value that contains the hours, minutes, seconds, and milliseconds set.

API version 34.0 to API version 53.0

If you call `Date.valueOf` with a `Datetime` object, the method converts `Datetime` to a valid `Date` without the time information, but the result depends on the manner in which the `Datetime` object was initialized. For example, if the `Datetime` object was initialized using `Datetime.valueOf(stringDate)`, the returned `Date` value contains time (hours) information. If the `Datetime` object is initialized using `Datetime.newInstance(year, month, day, hour, minute, second)` the returned `Date` value doesn't contain time information.

API version 54.0 and later

If you call `Date.valueOf` with a `Datetime` object, the method converts the object to a valid `Date` without the time information.

year()

Returns the year component of a Date

Signature

```
public Integer year()
```

Return Value

Type: Integer

Example

```
date myDate = date.newInstance(1988, 12, 17);
system.assertEquals(1988, myDate.year());
```