Chrono for Octave

version 0.3.1, January 2019

Andrew Janke

This manual is for Chrono, version 0.3.1.

Copyright © 2019 Andrew Janke

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the same conditions as for modified versions.

Short Contents

1 Introduction 1
2 Getting Started 2
3 Date Representation 3
4 Time Zones 4
5 Durations 6
6 Missing Functionality 7
7 Function Reference 8
8 Copying 26

Table of Contents

1 Introduction 1

2 Getting Started 2

3 Date Representation 3

3.1 `datetime` Class 3
3.1.1 `datenum` Compatibility 3

4 Time Zones 4

4.1 Defined Time Zones 5

5 Durations 6

5.1 `duration` Class 6
5.2 `calendarDuration` Class 6

6 Missing Functionality 7

7 Function Reference 8

7.1 Functions by Category 8
7.1.1 Date Representation 8
7.1.2 Durations 8
7.1.3 Uncategorized 9
7.2 Functions Alphabetically 9
7.2.1 calendarDuration 9
7.2.1.1 calendarDuration.calendarDuration 9
7.2.1.2 calendarDuration.isnat 10
7.2.1.3 calendarDuration.uminus 10
7.2.1.4 calendarDuration.plus 10
7.2.1.5 calendarDuration.times 10
7.2.1.6 calendarDuration.minus 10
7.2.1.7 calendarDuration.dispstrs 10
7.2.1.8 calendarDuration.isnan 10
7.2.2 calmonths 11
7.2.3 calyears 11
7.2.4 datetime 11
7.2.4.1 datetime.datetime 12
7.2.4.2 datetime.ofDatenum 12
7.2.4.3 datetime.ofDatestruct 12
7.2.4.4 datetime.NaT 12
7.2.4.5 datetime.posix2datenum 12

7.2.4.6 datetime.datenum2posix . . . 13
7.2.4.7 datetime.proxyKeys . . . 13
7.2.4.8 datetime.ymd . . . 13
7.2.4.9 datetime.hms . . . 13
7.2.4.10 datetime.ymdhms . . . 13
7.2.4.11 datetime.timeofday . . . 13
7.2.4.12 datetime.week . . . 14
7.2.4.13 datetime.dispstrs . . . 14
7.2.4.14 datetime.datestr . . . 14
7.2.4.15 datetime.datestrs . . . 14
7.2.4.16 datetime.datestruct . . . 14
7.2.4.17 datetime.posixtime . . . 14
7.2.4.18 datetime.datenum . . . 15
7.2.4.19 datetime.isnat . . . 15
7.2.4.20 datetime.isnan . . . 15
7.2.4.21 datetime.lt . . . 15
7.2.4.22 datetime.le . . . 15
7.2.4.23 datetime.ne . . . 15
7.2.4.24 datetime.eq . . . 16
7.2.4.25 datetime.ge . . . 16
7.2.4.26 datetime.gt . . . 16
7.2.4.27 datetime.plus . . . 16
7.2.4.28 datetime.minus . . . 16
7.2.4.29 datetime.diff . . . 16
7.2.4.30 datetime.isbetween . . . 17
7.2.4.31 datetime.linspace . . . 17
7.2.4.32 datetime.convertDatenumTimeZone . . . 17
7.2.5 days . . . 17
7.2.6 duration . . . 17
7.2.6.1 duration.duration . . . 18
7.2.6.2 duration.ofDays . . . 18
7.2.6.3 duration.ofDays . . . 18
7.2.6.4 duration.years . . . 18
7.2.6.5 duration.years . . . 18
7.2.6.6 duration.hours . . . 18
7.2.6.7 duration.hours . . . 18
7.2.6.8 duration.minutes . . . 19
7.2.6.9 duration.minutes . . . 19
7.2.6.10 duration.seconds . . . 19
7.2.6.11 duration.seconds . . . 19
7.2.6.12 duration.milliseconds . . . 19
7.2.6.13 duration.milliseconds . . . 19
7.2.6.14 duration.dispstrs . . . 19
7.2.6.15 duration.dispstrs . . . 19
7.2.6.16 duration.char . . . 19
7.2.6.17 duration.char . . . 19
7.2.6.18 duration.linspace . . . 19
7.2.6.19 duration.linspace . . . 20

7.2.7 hours . . . 20
7.2.8 isdatetime . . . 20
7.2.9 isduration . . . 20
7.2.10 localdate . . . 20
7.2.10.1 localdate.localdate . . . 21
7.2.10.2 localdate.NaT . . . 21
7.2.10.3 localdate.ymd . . . 21
7.2.10.4 localdate.dispstrs . . . 21
7.2.10.5 localdate.datestr . . . 21
7.2.10.6 localdate.datestrs . . . 21
7.2.10.7 localdate.datestruct . . . 22
7.2.10.8 localdate.posixtime . . . 22
7.2.10.9 localdate.datenum . . . 22
7.2.10.10 localdate.isnat . . . 22
7.2.10.11 localdate.isnan . . . 22
7.2.11 milliseconds . . . 22
7.2.12 minutes . . . 23
7.2.13 NaT . . . 23
7.2.14 octave.chrono.dummy_function . . . 23
7.2.15 octave.chrono.DummyClass . . . 23
7.2.15.1 octave.chrono.DummyClass.DummyClass . . . 23
7.2.15.2 octave.chrono.DummyClass.foo . . . 24
7.2.15.3 octave.chrono.DummyClass.bar . . . 24
7.2.16 seconds . . . 24
7.2.17 timezones . . . 24
7.2.18 years . . . 24

8 Copying . . . 26
8.1 Package Copyright . . . 26
8.2 Manual Copyright . . . 26

1 Introduction

> Time is an illusion. Lunchtime doubly so.
>
> —*Douglas Adams*

This is the manual for the Chrono package version 0.3.1 for GNU Octave.

This document is a work in progress. You are invited to help improve it and submit patches.

Chrono provides date/time functionality for Octave by supplying Matlab-compatible implementations for the `datetime`, `duration`, and `calendarDuration` classes, along with related functions.

Chrono's classes are designed to be convenient to use while still being efficient. The data representations used by Chrono are designed to be efficient and suitable for working with large-ish data sets. A "large-ish" data set is one that can have millions of elements or rows, but still fits in main computer memory. Chrono's main relational and arithmetic operations are all implemented using vectorized operations on primitive Octave data types.

Chrono was written by Andrew Janke `<floss@apjanke.net>`. Support can be found on the Chrono project GitHub page (`https://github.com/apjanke/octave-chrono`).

2 Getting Started

The easiest way to obtain Chrono is by using Octave's `pkg` package manager. To install the a development prerelease of Chrono, run this in Octave:

```
pkg install https://github.com/apjanke/octave-chrono/releases/download/v0.3.1/chrono-0
```

(Check the releases page at `https://github.com/apjanke/octave-chrono/releases` to find out what the actual latest release number is.)

For development, you can obtain the source code for Chrono from the project repo on GitHub at `https://github.com/apjanke/octave-chrono`. Upon first installation, run the `octave_chrono_make_local` script to build the octfiles so Chrono will work. Then add the `inst` directory in the repo to your Octave path.

3 Date Representation

Chrono provides the `datetime` class for representing points in time.

3.1 `datetime` Class

A `datetime` is an array object that represents points in time in the familiar Gregorian calendar.

This is an attempt to reproduce the functionality of Matlab's `datetime`. It also contains some Octave-specific extensions.

The underlying representation is that of a datenum (a `double` containing the number of days since the Matlab epoch), but encapsulating it in an object provides several benefits: friendly human-readable display, type safety, automatic type conversion, and time zone support. In addition to the underlying datenum array, a `datetime` inclues an optional `TimeZone` property indicating what time zone the datetimes are in.

3.1.1 `datenum` Compatibility

While the underlying data representation of `datetime` is compatible with (in fact, identical to) that of datenums, you cannot directly combine them via assignment, concatenation, or most arithmetic operations.

This is because of the signature of the `datetime` constructor. When combining objects and primitive types like `double`, the primitive type is promoted to an object by calling the other object's one-argument constructor on it. However, the one-argument numeric-input consstructor for `datetime` does not accept datenums: it interprets its input as datevecs instead. This is due to a design decision on Matlab's part; for compatibility, Octave does not alter that interface.

To combine `datetime`s with datenums, you can convert the datenums to `datetime`s by calling `datetime.ofDatenum` or `datetime(x, 'ConvertFrom', 'datenum')`, or you can convert the `datetime`s to datenums by accessing its `dnums` field with `x.dnums`.

Examples:

```
dt = datetime('2011-03-04')
dn = datenum('2017-01-01')
[dt dn]
    ⇒ error: datenum: expected date vector containing [YEAR, MONTH, DAY, HOUR, MINUTE
[dt datetime.ofDatenum(dn)]
    ⇒ 04-Mar-2011   01-Jan-2017
```

Also, if you have a zoned `datetime`, you can't combine it with a datenum, because datenums do not carry time zone information.

4 Time Zones

Chrono has support for representing dates in time zones and for converting between time zones.

A `datetime` may be "zoned" or "zoneless". A zoneless `datetime` does not have a time zone associated with it. This is represented by an empty `TimeZone` property on the `datetime` object. A zoneless `datetime` represents the local time in some unknown time zone, and assumes a continuous time scale (no DST shifts).

A zoned `datetime` is associated with a time zone. It is represented by having the time zone's IANA zone identifier (e.g. `'UTC'` or `'America/New_York'`) in its `TimeZone` property. A zoned `datetime` represents the local time in that time zone.

By default, the `datetime` constructor creates unzoned `datetime`s. To make a zoned `datetime`, either pass the `'TimeZone'` option to the constructor, or set the `TimeZone` property after object creation. Setting the `TimeZone` property on a zoneless `datetime` declares that it's a local time in that time zone. Setting the `TimeZone` property on a zoned `datetime` turns it back into a zoneless `datetime` without changing the local time it represents.

You can tell a zoned from a zoneless time zone in the object display because the time zone is included for zoned `datetime`s.

```
% Create an unzoned datetime
d = datetime('2011-03-04 06:00:00')
    ⇒  04-Mar-2011 06:00:00

% Create a zoned datetime
d_ny = datetime('2011-03-04 06:00:00', 'TimeZone', 'America/New_York')
    ⇒  04-Mar-2011 06:00:00 America/New_York
% This is equivalent
d_ny = datetime('2011-03-04 06:00:00');
d_ny.TimeZone = 'America/New_York'
    ⇒  04-Mar-2011 06:00:00 America/New_York

% Convert it to Chicago time
d_chi.TimeZone = 'America/Chicago'
    ⇒  04-Mar-2011 05:00:00 America/Chicago
```

When you combine two zoned `datetime`s via concatenation, assignment, or arithmetic, if their time zones differ, they are converted to the time zone of the left-hand input.

```
d_ny = datetime('2011-03-04 06:00:00', 'TimeZone', 'America/New_York')
d_la = datetime('2011-03-04 06:00:00', 'TimeZone', 'America/Los_Angeles')
d_la - d_ny
    ⇒ 03:00:00
```

You cannot combine a zoned and an unzoned `datetime`. This results in an error being raised.

> **Warning:** Normalization of "nonexistent" times (like between 02:00 and 03:00 on a "spring forward" DST change day) is not implemented yet. The results of converting a zoneless local time into a time zone where that local time did not exist are currently undefined.

4.1 Defined Time Zones

Chrono's time zone data is drawn from the IANA Time Zone Database (`https://www.iana.org/time-zones`), also known as the "Olson Database". Chrono includes a copy of this database in its distribution so it can work on Windows, which does not supply it like Unix systems do.

You can use the `timezones` function to list the time zones known to Chrono. These will be all the time zones in the IANA database on your system (for Linux and macOS) or in the IANA time zone database redistributed with Chrono (for Windows).

> **Note:** The IANA Time Zone Database only covers dates from about the year 1880 to 2038. Converting time zones for `datetime`s outside that range is currently unimplemented. (Chrono needs to add support for proleptic POSIX time zone rules, which are used to govern behavior outside that date range.)

5 Durations

5.1 `duration` Class

A `duration` represents a period of time in fixed-length seconds (or minutes, hours, or whatever you want to measure it in.)

A `duration` has a resolution of about a nanosecond for typical dates. The underlying representation is a `double` representing the number of days elapsed, similar to a datenum, except it's interpreted as relative to some other reference point you provide, instead of being relative to the Matlab/Octave epoch.

You can add or subtract a `duration` to a `datetime` to get another `datetime`. You can also add or subtract `durations` to each other.

5.2 `calendarDuration` Class

A `calendarDuration` represents a period of time in variable-length calendar components. For example, years and months can have varying numbers of days, and days in time zones with Daylight Saving Time have varying numbers of hours. A `calendarDuration` does arithmetic with "whole" calendar periods.

`calendarDuration`s and `duration`s cannot be directly combined, because they are not semantically equivalent. (This may be relaxed in the future to allow `duration`s to be interpreted as numbers of days when combined with `calendarDuration`s.)

```
d = datetime('2011-03-04 00:00:00')
    ⇒ 04-Mar-2011
cdur = calendarDuration(1, 3, 0)
    ⇒ 1y 3mo
d2 = d + cdur
    ⇒ 04-Jun-2012
```

6 Missing Functionality

Chrono is based on Matlab's date/time API and supports most of its major functionality. But not all of it is implemented yet. The missing parts are currently:

- POSIX time zone support for years outside the IANA time zone database coverage
- Week-of-year (ISO calendar) calculations
- Various `'ConvertFrom'` forms for `datetime` and `duration`
- Support for LDML formatting for `datetime`
- Various functions: `between`, `caldiff`, `dateshift`, `week`
- `isdst`, `isweekend`
- `calendarDuration.split`
- `duration.Format` support
- `UTCOffset` and `DSTOffset` fields in the output of `timezones()`
- Plotting support

It is the author's hope that all these will be implemented some day.

7 Function Reference

7.1 Functions by Category

7.1.1 Date Representation

Section 7.2.4 [datetime], page 11
Represents points in time using the Gregorian calendar.

Section 7.2.10 [localdate], page 20
Represents a complete day using the Gregorian calendar.

Section 7.2.8 [isdatetime], page 20
True if input is a 'datetime' array, false otherwise.

Section 7.2.13 [NaT], page 23
"Not-a-Time".

7.1.2 Durations

Section 7.2.1 [calendarDuration], page 9
Durations of time using variable-length calendar periods, such as days, months, and years, which may vary in length over time.

Section 7.2.2 [calmonths], page 11
Create a 'calendarDuration' that is a given number of calendar months long.

Section 7.2.3 [calyears], page 11
Construct a 'calendarDuration' a given number of years long.

Section 7.2.5 [days], page 17
Duration in days.

Section 7.2.6 [duration], page 17
Represents durations or periods of time as an amount of fixed-length time (i.e.

Section 7.2.7 [hours], page 20
Create a 'duration' X hours long, or get the hours in a 'duration' X.

Section 7.2.9 [isduration], page 20
True if input is a 'duration' array, false otherwise.

Section 7.2.11 [milliseconds], page 22
Create a 'duration' X milliseconds long, or get the milliseconds in a 'duration' X.

Section 7.2.12 [minutes], page 23
Create a 'duration' X hours long, or get the hours in a 'duration' X.

Section 7.2.16 [seconds], page 24
Create a 'duration' X seconds long, or get the seconds in a 'duration' X.

Section 7.2.17 [timezones], page 24
List all the time zones defined on this system.

Section 7.2.18 [years], page 24
Create a 'duration' X years long, or get the years in a 'duration' X.

7.1.3 Uncategorized

Section 7.2.14 [octave.chrono.dummy_function], page 23
A dummy function just for testing the doco tools.

Section 7.2.15 [octave.chrono.DummyClass], page 23
'DummyClass' is a do-nothing class just for testing the doco tools.

7.2 Functions Alphabetically

7.2.1 calendarDuration

`calendarDuration` [Class]
Durations of time using variable-length calendar periods, such as days, months, and years, which may vary in length over time. (For example, a calendar month may have 28, 30, or 31 days.)

`char Sign` [Instance Variable of `calendarDuration`]
The sign (1 or -1) of this duration, which indicates whether it is a positive or negative span of time.

`char Years` [Instance Variable of `calendarDuration`]
The number of whole calendar years in this duration. Must be integer-valued.

`char Months` [Instance Variable of `calendarDuration`]
The number of whole calendar months in this duration. Must be integer-valued.

`char Days` [Instance Variable of `calendarDuration`]
The number of whole calendar days in this duration. Must be integer-valued.

`char Hours` [Instance Variable of `calendarDuration`]
The number of whole hours in this duration. Must be integer-valued.

`char Minutes` [Instance Variable of `calendarDuration`]
The number of whole minutes in this duration. Must be integer-valued.

`char Seconds` [Instance Variable of `calendarDuration`]
The number of seconds in this duration. May contain fractional values.

`char Format` [Instance Variable of `calendarDuration`]
The format to display this `calendarDuration` in. Currently unsupported.

This is a single value that applies to the whole array.

7.2.1.1 calendarDuration.calendarDuration

obj = `calendarDuration` () [Constructor]
Constructs a new scalar `calendarDuration` of zero elapsed time.

obj = `calendarDuration` (*Y*, *M*, *D*) [Constructor]
obj = `calendarDuration` (*Y*, *M*, *D*, *H*, *MI*, *S*) [Constructor]
Constructs new `calendarDuration` arrays based on input values.

7.2.1.2 calendarDuration.isnat

`out = isnat (obj)` [Method]
True if input elements are NaT.

Returns logical array the same size as *obj*.

7.2.1.3 calendarDuration.uminus

`out = uminus (obj)` [Method]
Unary minus. Negates the sign of *obj*.

7.2.1.4 calendarDuration.plus

`out = plus (A, B)` [Method]
Addition: add two `calendarDuration`s.

All the calendar elements (properties) of the two inputs are added together. No normalization is done across the elements, aside from the normalization of NaNs.

If *B* is numeric, it is converted to a `calendarDuration` using `calendarDuration.ofDays`.

Returns a `calendarDuration`.

7.2.1.5 calendarDuration.times

`out = times (obj, B)` [Method]
Multiplication: Multiplies a `calendarDuration` by a numeric factor.

Returns a `calendarDuration`.

7.2.1.6 calendarDuration.minus

`out = times (A, B)` [Method]
Subtraction: Subtracts one `calendarDuration` from another.

Returns a `calendarDuration`.

7.2.1.7 calendarDuration.dispstrs

`out = dispstrs (obj)` [Method]
Get display strings for each element of *obj*.

Returns a cellstr the same size as *obj*.

7.2.1.8 calendarDuration.isnan

`out = isnan (obj)` [Method]
True if input elements are NaT. This is just an alias for `isnat`, provided for compatibility and polymorphic programming purposes.

Returns logical array the same size as *obj*.

7.2.2 calmonths

out = `calmonths (x)` [Function File]

Create a `calendarDuration` that is a given number of calendar months long.

Input *x* is a numeric array specifying the number of calendar months.

This is a shorthand alternative to calling the `calendarDuration` constructor with `calendarDuration(0, x, 0)`.

Returns a new `calendarDuration` object of the same size as *x*.

See Section 7.2.1 [calendarDuration], page 9.

7.2.3 calyears

out = `calyears (x)` [Function]

Construct a `calendarDuration` a given number of years long.

This is a shorthand for calling `calendarDuration(x, 0, 0)`.

See Section 7.2.1 [calendarDuration], page 9.

7.2.4 datetime

`datetime` [Class]

Represents points in time using the Gregorian calendar.

The underlying values are doubles representing the number of days since the Matlab epoch of "January 0, year 0". This has a precision of around nanoseconds for typical times.

A `datetime` array is an array of date/time values, with each element holding a complete date/time. The overall array may also have a TimeZone and a Format associated with it, which apply to all elements in the array.

This is an attempt to reproduce the functionality of Matlab's `datetime`. It also contains some Octave-specific extensions.

`double dnums` [Instance Variable of `datetime`]

The underlying datenums that represent the points in time. These are always in UTC.

This is a planar property: the size of `dnums` is the same size as the containing `datetime` array object.

`char TimeZone` [Instance Variable of `datetime`]

The time zone this `datetime` array is in. Empty if this does not have a time zone associated with it ("unzoned"). The name of an IANA time zone if this does.

Setting the `TimeZone` of a `datetime` array changes the time zone it is presented in for strings and broken-down times, but does not change the underlying UTC times that its elements represent.

`char Format` [Instance Variable of `datetime`]

The format to display this `datetime` in. Currently unsupported.

7.2.4.1 datetime.datetime

obj = `datetime` () [Constructor]
Constructs a new scalar `datetime` containing the current local time, with no time zone attached.

obj = `datetime` (*datevec*) [Constructor]
obj = `datetime` (*datestrs*) [Constructor]
obj = `datetime` (*in*, `'ConvertFrom'`, *inType*) [Constructor]
obj = `datetime` (*Y*, *M*, *D*, *H*, *MI*, *S*) [Constructor]
obj = `datetime` (*Y*, *M*, *D*, *H*, *MI*, *MS*) [Constructor]
obj = `datetime` (*...*, `'Format'`, *Format*, `'InputFormat'`, *InputFormat*, `'Locale'`, *InputLocale*, `'PivotYear'`, *PivotYear*, `'TimeZone'`, *TimeZone*) [Constructor]
Constructs a new `datetime` array based on input values.

7.2.4.2 datetime.ofDatenum

obj = `datetime.ofDatenum` (*dnums*) [Static Method]
Converts a datenum array to a datetime array.

Returns an unzoned `datetime` array of the same size as the input.

7.2.4.3 datetime.ofDatestruct

obj = `datetime.ofDatestruct` (*dstruct*) [Static Method]
Converts a datestruct to a datetime array.

A datestruct is a special struct format used by Chrono that has fields Year, Month, Day, Hour, Minute, and Second. It is not a standard Octave datatype.

Returns an unzoned `datetime` array.

7.2.4.4 datetime.NaT

out = `datetime.NaT` () [Static Method]
out = `datetime.NaT` (*sz*) [Static Method]
"Not-a-Time": Creates NaT-valued arrays.

Constructs a new `datetime` array of all `NaT` values of the given size. If no input *sz* is given, the result is a scalar `NaT`.

`NaT` is the `datetime` equivalent of `NaN`. It represents a missing or invalid value. `NaT` values never compare equal to, greater than, or less than any value, including other `NaT`s. Doing arithmetic with a `NaT` and any other value results in a `NaT`.

7.2.4.5 datetime.posix2datenum

dnums = `datetime.posix2datenum` (*pdates*) [Static Method]
Converts POSIX (Unix) times to datenums

Pdates (numeric) is an array of POSIX dates. A POSIX date is the number of seconds since January 1, 1970 UTC, excluding leap seconds. The output is implicitly in UTC.

7.2.4.6 datetime.datenum2posix

out = `datetime.datenum2posix` (*dnums*) [Static Method]

Converts Octave datenums to Unix dates.

The input datenums are assumed to be in UTC.

Returns a double, which may have fractional seconds.

7.2.4.7 datetime.proxyKeys

[*keysA*, *keysB*] = `proxyKeys` (`a`, *b*) [Method]

Computes proxy key values for two datetime arrays. Proxy keys are numeric values whose rows have the same equivalence relationships as the elements of the inputs.

This is primarily for Chrono's internal use; users will typically not need to call it or know how it works.

Returns two 2-D numeric matrices of size n-by-k, where n is the number of elements in the corresponding input.

7.2.4.8 datetime.ymd

[*y*, *m*, *d*] = `ymd` (*obj*) [Method]

Get the Year, Month, and Day components of *obj*.

For zoned `datetime`s, these will be local times in the associated time zone.

Returns double arrays the same size as `obj`.

7.2.4.9 datetime.hms

[*h*, *m*, *s*] = `hms` (*obj*) [Method]

Get the Hour, Minute, and Second components of a *obj*.

For zoned `datetime`s, these will be local times in the associated time zone.

Returns double arrays the same size as `obj`.

7.2.4.10 datetime.ymdhms

[*y*, *m*, *d*, *h*, *mi*, *s*] = `ymdhms` (*obj*) [Method]

Get the Year, Month, Day, Hour, Minute, and Second components of a *obj*.

For zoned `datetime`s, these will be local times in the associated time zone.

Returns double arrays the same size as `obj`.

7.2.4.11 datetime.timeofday

out = `timeofday` (*obj*) [Method]

Get the time of day (elapsed time since midnight).

For zoned `datetime`s, these will be local times in the associated time zone.

Returns a `duration` array the same size as `obj`.

7.2.4.12 datetime.week

`out = week (obj)` [Method]

Get the week of the year.

This method is unimplemented.

7.2.4.13 datetime.dispstrs

`out = dispstrs (obj)` [Method]

Get display strings for each element of *obj*.

Returns a cellstr the same size as *obj*.

7.2.4.14 datetime.datestr

`out = datestr (obj)` [Method]
`out = datestr (obj, ...)` [Method]

Format *obj* as date strings. Supports all arguments that core Octave's `datestr` does.

Returns date strings as a 2-D char array.

7.2.4.15 datetime.datestrs

`out = datestrs (obj)` [Method]
`out = datestrs (obj, ...)` [Method]

Format *obj* as date strings, returning cellstr. Supports all arguments that core Octave's `datestr` does.

Returns a cellstr array the same size as *obj*.

7.2.4.16 datetime.datestruct

`out = datestruct (obj)` [Method]

Converts this to a "datestruct" broken-down time structure.

A "datestruct" is a format of struct that Chrono came up with. It is a scalar struct with fields Year, Month, Day, Hour, Minute, and Second, each containing a double array the same size as the date array it represents.

The values in the returned broken-down time are those of the local time in this' defined time zone, if it has one.

Returns a struct with fields Year, Month, Day, Hour, Minute, and Second. Each field contains a double array of the same size as this.

7.2.4.17 datetime.posixtime

`out = posixtime (obj)` [Method]

Converts this to POSIX time values (seconds since the Unix epoch)

Converts this to POSIX time values that represent the same time. The returned values will be doubles that may include fractional second values. POSIX times are, by definition, in UTC.

Returns double array of same size as this.

7.2.4.18 datetime.datenum

out = `datenum` (*obj*) [Method]
Convert this to datenums that represent the same local time

Returns double array of same size as this.

7.2.4.19 datetime.isnat

out = `isnat` (*obj*) [Method]
True if input elements are NaT.

Returns logical array the same size as *obj*.

7.2.4.20 datetime.isnan

out = `isnan` (*obj*) [Method]
True if input elements are NaT. This is an alias for `isnat` to support type compatibility and polymorphic programming.

Returns logical array the same size as *obj*.

7.2.4.21 datetime.lt

out = `lt` (*A*, *B*) [Method]
True if *A* is less than *B*. This defines the `<` operator for `datetime`s.

Inputs are implicitly converted to `datetime` using the one-arg constructor or conversion method.

Returns logical array the same size as *obj*.

7.2.4.22 datetime.le

out = `le` (*A*, *B*) [Method]
True if *A* is less than or equal to*B*. This defines the `<=` operator for `datetime`s.

Inputs are implicitly converted to `datetime` using the one-arg constructor or conversion method.

Returns logical array the same size as *obj*.

7.2.4.23 datetime.ne

out = `ne` (*A*, *B*) [Method]
True if *A* is not equal to *B*. This defines the `!=` operator for `datetime`s.

Inputs are implicitly converted to `datetime` using the one-arg constructor or conversion method.

Returns logical array the same size as *obj*.

7.2.4.24 datetime.eq

out = `eq` (*A*, *B*) [Method]

True if *A* is equal to *B*. This defines the `==` operator for `datetime`s.

Inputs are implicitly converted to `datetime` using the one-arg constructor or conversion method.

Returns logical array the same size as *obj*.

7.2.4.25 datetime.ge

out = `ge` (*A*, *B*) [Method]

True if *A* is greater than or equal to *B*. This defines the `>=` operator for `datetime`s.

Inputs are implicitly converted to `datetime` using the one-arg constructor or conversion method.

Returns logical array the same size as *obj*.

7.2.4.26 datetime.gt

out = `gt` (*A*, *B*) [Method]

True if *A* is greater than *B*. This defines the `>` operator for `datetime`s.

Inputs are implicitly converted to `datetime` using the one-arg constructor or conversion method.

Returns logical array the same size as *obj*.

7.2.4.27 datetime.plus

out = `plus` (*A*, *B*) [Method]

Addition (`+` operator). Adds a `duration`, `calendarDuration`, or numeric *B* to a `datetime` *A*.

Numeric *B* inputs are implicitly converted to `duration` using `duration.ofDays`.

Returns `datetime` array the same size as *A*.

7.2.4.28 datetime.minus

out = `minus` (*A*, *B*) [Method]

Subtraction (`-` operator). Subtracts a `duration`, `calendarDuration` or numeric *B* from a `datetime` *A*, or subtracts two `datetime`s from each other.

If both inputs are `datetime`, then the output is a `duration`. Otherwise, the output is a `datetime`.

Numeric *B* inputs are implicitly converted to `duration` using `duration.ofDays`.

Returns an array the same size as *A*.

7.2.4.29 datetime.diff

out = `diff` (*obj*) [Method]

Differences between elements.

Computes the difference between each successive element in *obj*, as a `duration`.

Returns a `duration` array the same size as *obj*.

7.2.4.30 datetime.isbetween

`out = isbetween (obj, lower, upper)` [Method]

Tests whether the elements of *obj* are between *lower* and *upper*.

All inputs are implicitly converted to `datetime` arrays, and are subject to scalar expansion.

Returns a logical array the same size as the scalar expansion of the inputs.

7.2.4.31 datetime.linspace

`out = linspace (from, to, n)` [Method]

Linearly-spaced values in date/time space.

Constructs a vector of `datetime`s that represent linearly spaced points starting at *from* and going up to *to*, with *n* points in the vector.

from and *to* are implicitly converted to `datetime`s.

n is how many points to use. If omitted, defaults to 100.

Returns an *n*-long `datetime` vector.

7.2.4.32 datetime.convertDatenumTimeZone

`out = datetime.convertDatenumTimeZone (dnum, fromZoneId, toZoneId)` [Static Method]

Convert a datenum from one time zone to another.

dnum is a datenum array to convert.

fromZoneId is a charvec containing the IANA Time Zone identifier for the time zone to convert from.

toZoneId is a charvec containing the IANA Time Zone identifier for the time zone to convert to.

Returns a datenum array the same size as *dnum*.

7.2.5 days

`out = days (x)` [Function]

Duration in days.

If *x* is numeric, then *out* is a `duration` array in units of fixed-length 24-hour days, with the same size as *x*.

If *x* is a `duration`, then returns a `double` array the same size as *x* indicating the number of fixed-length days that each duration is.

7.2.6 duration

`duration` [Class]

Represents durations or periods of time as an amount of fixed-length time (i.e. fixed-length seconds). It does not care about calendar things like months and days that vary in length over time.

This is an attempt to reproduce the functionality of Matlab's `duration`. It also contains some Octave-specific extensions.

`double days` [Instance Variable of `duration`]
The underlying datenums that represent the durations, as number of (whole and fractional) days. These are uniform 24-hour days, not calendar days.

This is a planar property: the size of `days` is the same size as the containing `duration` array object.

`char Format` [Instance Variable of `duration`]
The format to display this `duration` in. Currently unsupported.

7.2.6.1 duration.duration

obj = `duration ()` [Constructor]
Constructs a new scalar `duration` of zero elapsed time.

obj = `duration (`*durationstrs*`)` [Constructor]
obj = `duration (`*durationstrs*`, 'InputFormat', `*InputFormat*`)` [Constructor]
obj = `duration (`*H*`, `*MI*`, `*S*`)` [Constructor]
obj = `duration (`*H*`, `*MI*`, `*S*`, `*MS*`)` [Constructor]
Constructs a new `duration` array based on input values.

7.2.6.2 duration.ofDays

7.2.6.3 duration.ofDays

obj = `duration.ofDays (`*dnums*`)` [Static Method]
Converts a double array representing durations in whole and fractional days to a `duration` array. This is the method that is used for implicit conversion of numerics in many cases.

Returns a `duration` array of the same size as the input.

7.2.6.4 duration.years

7.2.6.5 duration.years

out = `years (`*obj*`)` [Method]
Equivalent number of years.

Gets the number of fixed-length 365.2425-day years that is equivalent to this duration.

Returns double array the same size as *obj*.

7.2.6.6 duration.hours

7.2.6.7 duration.hours

out = `hours (`*obj*`)` [Method]
Equivalent number of hours.

Gets the number of fixed-length 60-minute hours that is equivalent to this duration.

Returns double array the same size as *obj*.

7.2.6.8 duration.minutes

7.2.6.9 duration.minutes

`out = minutes (obj)` [Method]

Equivalent number of minutes.

Gets the number of fixed-length 60-second minutes that is equivalent to this duration.

Returns double array the same size as *obj*.

7.2.6.10 duration.seconds

7.2.6.11 duration.seconds

`out = seconds (obj)` [Method]

Equivalent number of seconds.

Gets the number of seconds that is equivalent to this duration.

Returns double array the same size as *obj*.

7.2.6.12 duration.milliseconds

7.2.6.13 duration.milliseconds

`out = milliseconds (obj)` [Method]

Equivalent number of milliseconds.

Gets the number of milliseconds that is equivalent to this duration.

Returns double array the same size as *obj*.

7.2.6.14 duration.dispstrs

7.2.6.15 duration.dispstrs

`out = duration (obj)` [Method]

Get display strings for each element of *obj*.

Returns a cellstr the same size as *obj*.

7.2.6.16 duration.char

7.2.6.17 duration.char

`out = char (obj)` [Method]

Convert to char. The contents of the strings will be the same as returned by `dispstrs`.

This is primarily a convenience method for use on scalar *obj*s.

Returns a 2-D char array with one row per element in *obj*.

7.2.6.18 duration.linspace

7.2.6.19 duration.linspace

`out = linspace (from, to, n)` [Method]

Linearly-spaced values in time duration space.

Constructs a vector of `duration`s that represent linearly spaced points starting at *from* and going up to *to*, with *n* points in the vector.

from and *to* are implicitly converted to `duration`s.

n is how many points to use. If omitted, defaults to 100.

Returns an *n*-long `datetime` vector.

7.2.7 hours

`out = hours (x)` [Function File]

Create a `duration` *x* hours long, or get the hours in a `duration` *x*.

If input is numeric, returns a `duration` array that is that many hours in time.

If input is a `duration`, converts the `duration` to a number of hours.

Returns an array the same size as *x*.

7.2.8 isdatetime

`tf = isdatetime (x)` [Function]

True if input is a `datetime` array, false otherwise.

Returns a logical array the same size as *x*.

7.2.9 isduration

`tf = isduration (x)` [Function]

True if input is a `duration` array, false otherwise.

Returns a logical array the same size as *x*.

7.2.10 localdate

`localdate` [Class]

Represents a complete day using the Gregorian calendar.

This class is useful for indexing daily-granularity data or representing time periods that cover an entire day in local time somewhere. The major purpose of this class is "type safety", to prevent time-of-day values from sneaking in to data sets that should be daily only. As a secondary benefit, this uses less memory than datetimes.

`double dnums` [Instance Variable of `localdate`]

The underlying datenum values that represent the days.

These are doubles, but they are restricted to be integer-valued, so they represent complete days, with no time-of-day component.

`char Format` [Instance Variable of `localdate`]

The format to display this `localdate` in. Currently unsupported.

7.2.10.1 localdate.localdate

obj = `localdate` () [Constructor]
Constructs a new scalar `localdate` containing the current local date.

obj = `localdate` (*datenums*) [Constructor]
obj = `localdate` (*datestrs*) [Constructor]
obj = `localdate` (*Y*, *M*, *D*) [Constructor]
obj = `localdate` (*...*, `'Format'`, *Format*) [Constructor]
Constructs a new `localdate` array based on input values.

7.2.10.2 localdate.NaT

out = `localdate.NaT` () [Static Method]
out = `localdate.NaT` (*sz*) [Static Method]
"Not-a-Time": Creates NaT-valued arrays.

Constructs a new `datetime` array of all `NaT` values of the given size. If no input *sz* is given, the result is a scalar `NaT`.

`NaT` is the `datetime` equivalent of `NaN`. It represents a missing or invalid value. `NaT` values never compare equal to, greater than, or less than any value, including other `NaT`s. Doing arithmetic with a `NaT` and any other value results in a `NaT`.

This static method is provided because the global `NaT` function creates `datetime`s, not `localdate`s

7.2.10.3 localdate.ymd

[*y*, *m*, *d*] = `ymd` (*obj*) [Method]
Get the Year, Month, and Day components of *obj*.

Returns double arrays the same size as `obj`.

7.2.10.4 localdate.dispstrs

out = `dispstrs` (*obj*) [Method]
Get display strings for each element of *obj*.

Returns a cellstr the same size as *obj*.

7.2.10.5 localdate.datestr

out = `datestr` (*obj*) [Method]
out = `datestr` (*obj*, *...*) [Method]
Format *obj* as date strings. Supports all arguments that core Octave's `datestr` does.

Returns date strings as a 2-D char array.

7.2.10.6 localdate.datestrs

out = `datestrs` (*obj*) [Method]
out = `datestrs` (*obj*, *...*) [Method]
Format *obj* as date strings, returning cellstr. Supports all arguments that core Octave's `datestr` does.

Returns a cellstr array the same size as *obj*.

7.2.10.7 localdate.datestruct

out = `datestruct` (*obj*) [Method]

Converts this to a "datestruct" broken-down time structure.

A "datestruct" is a format of struct that Chrono came up with. It is a scalar struct with fields Year, Month, and Day, each containing a double array the same size as the date array it represents. This format differs from the "datestruct" used by `datetime` in that it lacks Hour, Minute, and Second components. This is done for efficiency.

The values in the returned broken-down time are those of the local time in this' defined time zone, if it has one.

Returns a struct with fields Year, Month, and Day. Each field contains a double array of the same size as this.

7.2.10.8 localdate.posixtime

out = `posixtime` (*obj*) [Method]

Converts this to POSIX time values for midnight of *obj*'s days.

Converts this to POSIX time values that represent the same date. The returned values will be doubles that will not include fractional second values. The times returned are those of midnight UTC on *obj*'s days.

Returns double array of same size as this.

7.2.10.9 localdate.datenum

out = `datenum` (*obj*) [Method]

Convert this to datenums that represent midnight on *obj*'s days.

Returns double array of same size as this.

7.2.10.10 localdate.isnat

out = `isnat` (*obj*) [Method]

True if input elements are NaT.

Returns logical array the same size as *obj*.

7.2.10.11 localdate.isnan

out = `isnan` (*obj*) [Method]

True if input elements are NaT. This is an alias for `isnat` to support type compatibility and polymorphic programming.

Returns logical array the same size as *obj*.

7.2.11 milliseconds

out = `milliseconds` (*x*) [Function File]

Create a `duration` *x* milliseconds long, or get the milliseconds in a `duration` *x*.

If input is numeric, returns a `duration` array that is that many milliseconds in time.

If input is a `duration`, converts the `duration` to a number of milliseconds.

Returns an array the same size as *x*.

7.2.12 minutes

out = `hours` (*x*) [Function File]
Create a `duration` *x* hours long, or get the hours in a `duration` *x*.

7.2.13 NaT

out = `NaT` () [Function]
out = `NaT` (*sz*) [Function]
"Not-a-Time". Creates NaT-valued arrays.

Constructs a new `datetime` array of all `NaT` values of the given size. If no input *sz* is given, the result is a scalar `NaT`.

`NaT` is the `datetime` equivalent of `NaN`. It represents a missing or invalid value. `NaT` values never compare equal to, greater than, or less than any value, including other `NaT`s. Doing arithmetic with a `NaT` and any other value results in a `NaT`.

7.2.14 octave.chrono.dummy_function

out = `dummy_function` (*x*) [Function]
A dummy function just for testing the doco tools.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur ullamcorper pulvinar ligula, sit amet accumsan turpis dapibus at. Ut sit amet quam orci. Donec vel mauris elementum massa pretium tincidunt.

7.2.15 octave.chrono.DummyClass

`DummyClass` [Class]
`DummyClass` is a do-nothing class just for testing the doco tools.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur ullamcorper pulvinar ligula, sit amet accumsan turpis dapibus at. Ut sit amet quam orci. Donec vel mauris elementum massa pretium tincidunt.

`double x` [Instance Variable of `DummyClass`]
A x. Has no semantics.

`double y` [Instance Variable of `DummyClass`]
A y. Has no semantics.

7.2.15.1 octave.chrono.DummyClass.DummyClass

obj = `octave.chrono.DummyClass` () [Constructor]
Constructs a new scalar `DummyClass` with default values.

obj = `octave.chrono.DummyClass` (*x*, *y*) [Constructor]
Constructs a new `DummyClass` with the specified values.

7.2.15.2 octave.chrono.DummyClass.foo

out = `foo` (*obj*) [Method]

Computes a foo value.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur ullamcorper pulvinar ligula, sit amet accumsan turpis dapibus at. Ut sit amet quam orci. Donec vel mauris elementum massa pretium tincidunt.

7.2.15.3 octave.chrono.DummyClass.bar

out = `bar` (*obj*) [Method]

Computes a bar value.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur ullamcorper pulvinar ligula, sit amet accumsan turpis dapibus at. Ut sit amet quam orci. Donec vel mauris elementum massa pretium tincidunt.

7.2.16 seconds

out = `seconds` (*x*) [Function File]

Create a `duration` *x* seconds long, or get the seconds in a `duration` *x*.

If input is numeric, returns a `duration` array that is that many seconds in time.

If input is a `duration`, converts the `duration` to a number of seconds.

Returns an array the same size as *x*.

7.2.17 timezones

out = `timezones` () [Function]

out = `timezones` (*area*) [Function]

List all the time zones defined on this system.

This lists all the time zones that are defined in the IANA time zone database used by this Octave. (On Linux and macOS, that will generally be the system time zone database from `/usr/share/zoneinfo`. On Windows, it will be the database redistributed with the Chrono package.

If the return is captured, the output is returned as a table if your Octave has table support, or a struct if it does not. It will have fields/variables containing column vectors:

`Name` The IANA zone name, as cellstr.

`Area` The geographical area the zone is in, as cellstr.

Compatibility note: Matlab also includes UTCOffset and DSTOffset fields in the output; these are currently unimplemented.

7.2.18 years

out = `years` (*x*) [Function File]

Create a `duration` *x* years long, or get the years in a `duration` *x*.

If input is numeric, returns a `duration` array in units of fixed-length years of 365.2425 days each.

If input is a `duration`, converts the `duration` to a number of fixed-length years as double.

Note: `years` creates fixed-length years, which may not be what you want. To create a duration of calendar years (which account for actual leap days), use `calyears`.

See Section 7.2.3 [calyears], page 11.

8 Copying

8.1 Package Copyright

Chrono for Octave is covered by the GNU GPLv3, the Unicode License, and Public Domain.

All the code in the package is GNU GPLv3.

The IANA Time Zone Database redistributed with the package is Public Domain.

The Windows Zones file redistributed with the package is covered by the Unicode License (`http://www.unicode.org/copyright.html`).

8.2 Manual Copyright

This manual is for Chrono, version 0.3.1.

Copyright © 2019 Andrew Janke

> Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.
>
> Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.
>
> Permission is granted to copy and distribute translations of this manual into another language, under the same conditions as for modified versions.