datetime-fortran

ci passing issues 7 open

Date and time manipulation for modern Fortran. The fundamental time step is one millisecond.

Getting started

First, get the code by cloning this repo:

```
git clone https://github.com/wavebitscientific/datetime-fortran
cd datetime-fortran
```

or by downloading a release tarball. The latter is recommended if you want to build datetime-fortran with autotools and make.

You can build datetime-fortran with FPM, autotools, CMake, or by simply including the source file "src/datetime_module.f90" in your project. NOTE: Windows additionally requires "src/strptime.cpp" in your project.

FPM

As of version 1.7.0, datetime-fortran can be built using the Fortran Package Manager. Follow the directions on that page to install FPM if you haven't already.

To build, type:

```
fpm build
```

binary artifacts are created in:

- Library and module files: `build/gfortran_debug/library`
- test executables: `build/gfortran_debug/tests`.

To run the tests, type:

```
fpm test
```

Autotools

```
./configure
make check
make install
```

If you cloned the repo instead of downloading a release tarball, run `autoreconf -i` first to generate the `configure` script.

CMake

```
cmake -B build
cmake --build build
```

binary artifacts are created in:

- Library: `build/lib/libdatetime`
- module: `build/include/datetime.mod`
- test executable: `build/bin/datetime_tests`

optionally, to install (replace "~/mylibs" with your desired install directory):

```
cmake -B build -DCMAKE_INSTALL_PREFIX=~/mylibs
cmake --install build
```

optionally, to run self-tests:

```
cd build
ctest -V
```

Use

Start using datetime-fortran in your code by importing derived types from the module:

```
use datetime_module, only: datetime, timedelta, clock
```

See some basic examples here.

API

- Derived Types
  - *datetime*
    - *getYear*
    - *getMonth*
    - *getDay*
    - *getHour*
    - *getMinute*
    - *getSecond*
    - *getMillisecond*
    - *isocalendar*
    - *isoformat*
    - *isValid*
    - *now*
    - *secondsSinceEpoch*
    - *strftime*
    - *tm*
    - *tzOffset*
    - *utc*

        - *weekday*
        - *weekdayLong*
        - *weekdayShort*
        - *yearday*
    - *timedelta*
        - *total_seconds*
    - *clock*
        - *reset*
        - *tick*
    - *tm_struct*
- Overloaded operators
    - Arithmetic operators
    - Comparison operators
- Public procedures
    - *c_strftime*
    - *c_strptime*
    - *date2num*
    - *datetimeRange*
    - *daysInMonth*
    - *daysInYear*
    - *isLeapYear*
    - *num2date*
    - *strptime*
    - *tm2date*

Derived Types

datetime-fortran library provides the following derived types: *datetime*, *timedelta*, *clock* and *tm_struct*.

datetime

Main date and time object, defined as:

```fortran
type :: datetime

  !! Main datetime class for date and time representation.

  private

  integer :: year        = 1 !! year [1-HUGE(year)]
  integer :: month       = 1 !! month in year [1-12]
  integer :: day         = 1 !! day in month [1-31]
  integer :: hour        = 0 !! hour in day [0-23]
  integer :: minute      = 0 !! minute in hour [0-59]
  integer :: second      = 0 !! second in minute [0-59]
  integer :: millisecond = 0 !! milliseconds in second [0-999]
```

```
real(kind=real64) :: tz = 0 !! timezone offset from UTC [hours]

contains

! getter functions
procedure,pass(self),public :: getYear
procedure,pass(self),public :: getMonth
procedure,pass(self),public :: getDay
procedure,pass(self),public :: getHour
procedure,pass(self),public :: getMinute
procedure,pass(self),public :: getSecond
procedure,pass(self),public :: getMillisecond
procedure,pass(self),public :: getTz

! public methods
procedure,pass(self),public :: isocalendar
procedure,pass(self),public :: isoformat
procedure,pass(self),public :: isValid
procedure,nopass,    public :: now
procedure,pass(self),public :: secondsSinceEpoch
procedure,pass(self),public :: strftime
procedure,pass(self),public :: tm
procedure,pass(self),public :: tzOffset
procedure,pass(self),public :: utc
procedure,pass(self),public :: weekday
procedure,pass(self),public :: isoweekday
procedure,pass(self),public :: weekdayLong
procedure,pass(self),public :: isoweekdayLong
procedure,pass(self),public :: weekdayShort
procedure,pass(self),public :: isoweekdayShort
procedure,pass(self),public :: yearday

! private methods
procedure,pass(self),private :: addMilliseconds
procedure,pass(self),private :: addSeconds
procedure,pass(self),private :: addMinutes
procedure,pass(self),private :: addHours
procedure,pass(self),private :: addDays

! operator overloading procedures
procedure,pass(d0),private :: datetime_plus_timedelta
procedure,pass(d0),private :: timedelta_plus_datetime
procedure,pass(d0),private :: datetime_minus_datetime
procedure,pass(d0),private :: datetime_minus_timedelta
procedure,pass(d0),private :: eq
```

```
  procedure,pass(d0),private :: neq
  procedure,pass(d0),private :: gt
  procedure,pass(d0),private :: ge
  procedure,pass(d0),private :: lt
  procedure,pass(d0),private :: le

  generic :: operator(+)  => datetime_plus_timedelta,&
                             timedelta_plus_datetime
  generic :: operator(-)  => datetime_minus_datetime,&
                             datetime_minus_timedelta
  generic :: operator(==) => eq
  generic :: operator(/=) => neq
  generic :: operator(>)  => gt
  generic :: operator(>=) => ge
  generic :: operator(<)  => lt
  generic :: operator(<=) => le

endtype datetime
```

datetime components are initialized by default, so all arguments are optional. Arguments may be provided as positional arguments, in the order of their declaration, or as keyword arguments, in any order. If both positional and keyword arguments are used, no positional arguments may appear after a keyword argument.

Example usage

```
use datetime_module, only:datetime

type(datetime) :: a

! Initialize as default:
a = datetime() ! 0001-01-01 00:00:00

! Components can be specified by position:
a = datetime(1984, 12, 10) ! 1984-12-10 00:00:00

! Or by keyword:
a = datetime(month=1, day=1, year=1970) ! 1970-01-01 00:00:00

! Or combined:
a = datetime(2013, 2, minute=23, day=5) ! 2013-02-05 00:23:00

! With timezone offset:
a = datetime(2013, 2, minute=23, day=5, tz=-4) ! 2013-02-05 00:23:00 -0400
```

```
! Do not use positional after keyword arguments:
a = datetime(year=2013, 2, minute=23, day=5) ! ILLEGAL
```

Note that the current implementation of *datetime* does not support daylight saving time (DST) information.

See also

- *timedelta*
- *tm_struct*

Back to top

getYear

```
pure elemental integer function getYear(self)
  class(datetime),intent(in) :: self
```

Returns the year of a `datetime` instance.

Back to top

getMonth

```
pure elemental integer function getMonth(self)
  class(datetime),intent(in) :: self
```

Returns the month of a `datetime` instance.

Back to top

getDay

```
pure elemental integer function getDay(self)
  class(datetime),intent(in) :: self
```

Returns the day of a `datetime` instance.

Back to top

getHour

```
pure elemental integer function getHour(self)
  class(datetime),intent(in) :: self
```

Returns the hour of a `datetime` instance.

Back to top

getMinute

```
pure elemental integer function getMinute(self)
  class(datetime),intent(in) :: self
```

Returns the minute of a `datetime` instance.

Back to top

getSecond

```
pure elemental integer function getSecond(self)
  class(datetime),intent(in) :: self
```

Returns the second of a `datetime` instance.

Back to top

getMillisecond

```
pure elemental integer function getMillisecond(self)
  class(datetime),intent(in) :: self
```

Returns the millisecond of a `datetime` instance.

Back to top

isocalendar

```
function isocalendar(self)
  class(datetime),intent(in) :: self
  integer,dimension(3)       :: isocalendar
```

Returns an array of 3 integers: year, week number, and week day, as defined by ISO 8601 week date. The ISO calendar is a widely used variant of the Gregorian calendar. The ISO year consists of 52 or 53 full weeks. A week starts on a Monday (1) and ends on a Sunday (7). The first week of an ISO year is the first (Gregorian) calendar week of a year containing a Thursday. This is called week number 1, and the ISO year of that Thursday is the same as its Gregorian year.

datetime%isocalendar() is equivalent to Python's *datetime.datetime.isocalendar()*.

Example usage

```
use datetime_module,only:datetime

type(datetime) :: a

a = datetime(2013,1,1)
print *, a % isocalendar() ! Prints: 2013  1  2
```

See also

- *weekday*

Back to top

isoformat

```
pure elemental character(len=23) function isoformat(self,sep)
  class(datetime), intent(in)          :: self
  character(len=1),intent(in),optional :: sep
```

Returns a character string of length 23 that contains date and time in ISO 8601 format.

datetime%isoformat() is equivalent to Python's *datetime.datetime.isoformat()*, with the only difference being that *datetime%isoformat()* returns the milliseconds at the end of the string, where as *datetime.datetime.isoformat()* returns microseconds.

Arguments `sep` is an optional argument that specifies which character of length 1 will separate date and time entries. If ommited, defaults to `T`.

Example usage

```
use datetime_module,only:datetime

type(datetime) :: a

a = datetime(1984,12,10,13,5,0)

! Without arguments:
print *, a % isoformat() ! Prints 1984-12-10T13:05:00.000

! With a specified separator:
print *, a % isoformat(' ') ! Prints 1984-12-10 13:05:00.000
```

See also Back to top

isValid

```
pure elemental logical function isValid(self)
  class(datetime),intent(in) :: self
```

Returns `.true.` if all *datetime* instance components have valid values, and .false. otherwise. Components have valid values if they are within the range indicated in *datetime* derived type description.

Useful for debugging and validating user input.

Example usage

```
use datetime_module,only:datetime

type(datetime) :: a

a = datetime(1984,12,10,13,5,0)

print *, a % isValid()! .true.

a = datetime(1984,13,10,13,5,0)

print *, a % isValid() ! .false.
```

See also Back to top

now

```
type(datetime) function now(self)
  class(datetime),intent(in) :: self
```

Returns the *datetime* instance representing the current machine time. Does not support timezones.

Return value `self` A `datetime` instance with current machine time.

Example usage

```
use datetime_module,only:datetime

type(datetime) :: a

a = a % now() ! Assigns current machine time to a
```

Back to top

secondsSinceEpoch

```
integer function secondsSinceEpoch(self)
  class(datetime),intent(in) :: self
```

Returns an integer number of seconds since the UNIX Epoch, `1970-01-01 00:00:00 +0000` (UTC).

Return value `secondsSinceEpoch` An `integer` scalar containing number of seconds since UNIX Epoch.

Example usage

```
use datetime_module,only:datetime

type(datetime) :: a

! Initialize:
a = datetime(2013,1,1)

print *, a%secondsSinceEpoch()
```

Back to top

strftime

```
character(len=maxstrlen) function strftime(self,format)
  class(datetime), intent(in) :: self
  character(len=*),intent(in) :: format
```

A *datetime*-bound method that serves as a wrapper around the C routine *strftime*. `datetime%strftime` takes only the format string as argument, and returns the character string representation of the time information contained in the datetime instance. Thus, this function takes care of the conversion to `tm_struct` and calling the raw C *strftime*. Because Fortran does not allow assumed-length character strings as the type of the function result, a fixed length of `MAXSTRLEN` is used. `MAXSTRLEN` is currently set to `99`. It is assumed that the desired time string is shorter than this value. Any resulting string shorter than `MAXSTRLEN` is padded with spaces, so it is best to trim the result using the `TRIM` intrinsic function (see the usage example below). This *datetime*-bound method is available since version `0.3.0`.

Arguments `format` A character string describing the desired format of date and time. Same as the format for the raw C *strftime*.

Return value A `character(len=maxstrlen)` representation of *datetime* using `format`.

Example usage

```
use datetime_module

type(datetime)  :: a

a = a % now()
print *, a % isoformat()

print *, trim(a % strftime("%Y %B %d"))
```

See also

- *c_strftime*

Back to top

tm

```
pure elemental type(tm_struct) function tm(self)
  CLASS(datetime),intent(in) :: self
```

Returns a *tm_struct* instance that matches the time and date information in the caller *datetime* instance.

Example usage

```
use datetime_module,only:datetime

type(datetime)  :: a
type(tm_struct) :: tm

! Initialize:
a = datetime(2013,1,1)

! Get tm_struct from datetime:
tm = a % tm()
```

See also

- *tm_struct*

Back to top

tzOffset

```
pure elemental character(len=5) function tzOffset(self)
  class(datetime),intent(in) :: self
```

Given a *datetime* instance, returns a character string with timezone offset in hours from UTC (Coordinated Universal Time), in format `+hhmm` or `-hhmm`, depending on the sign, where `hh` are hours and `mm` are minutes.

Arguments None.

Return value `tzOffset` A `character(len=5)` in the form `+hhmm` or `-hhmm`, depending on the sign.

Example usage

```
use datetime_module,only:datetime

type(datetime)  :: a
type(tm_struct) :: tm

! Initialize a datetime instance with timezone offset of -4.75 hours:
a = datetime(2013,1,1,tz=-4.75)

! Write tzOffset on screen:
print *, a % tzOffset ! -0445 (offset of 4 hours and 45 minutes)
```

Back to top

utc

```
pure elemental type(datetime) function utc(self)
  class(datetime),intent(in) :: self
```

Returns the datetime instance at Coordinated Universal Time (UTC).

Return value `utc` A `datetime` instance with at UTC (tz $= 0$).

Example usage

```
use datetime_module,only:datetime

type(datetime)  :: a
type(tm_struct) :: tm

! Initialize a datetime instance with timezone offset of -4.75 hours:
a = datetime(2013,1,1,tz=-4.75)

print *, a % isoformat() // a % tzOffset() ! 2013-01-01T00:00:00.000-0445

! Convert a to UTC:
a = a % utc()

print *, a % isoformat() // a % tzOffset() ! 2013-01-01T04:45:00.000+0000
```

See also

- *tzOffset*

Back to top

weekday

```
pure elemental integer function weekday(self)
  class(datetime),intent(in) :: self
```

A *datetime*-bound method to calculate day of the week using Zeller's congruence. Returns an integer scalar in the range of [0-6], starting from Sunday.

Example usage

```
use datetime_module,only:datetime

type(datetime)  :: a

! Initialize:
a = datetime(2013,1,1)

print *, a % weekday() ! 2
```

See also

- *weekdayLong*
- *weekdayShort*

Back to top

weekdayLong

```
pure elemental character(len=9) function weekdayLong(self)
  class(datetime),intent(in) :: self
```

Returns the full name of the day of the week.

Example usage

```
use datetime_module,only:datetime

type(datetime)  :: a

! Initialize:
a = datetime(2013,1,1)

print *, a % weekdayLong() ! Tuesday
```

See also

- *weekday*
- *weekdayShort*

Back to top

weekdayShort

```
pure elemental character(len=3) function weekdayShort(self)
  class(datetime),intent(in) :: self
```

Returns the abbreviated (e.g. Mon) name of the day of the week.

Example usage

```
use datetime_module,only:datetime

type(datetime)  :: a

! Initialize:
a = datetime(2013,1,1)

print *, a % weekdayShort() ! Tue
```

See also

- *weekday*
- *weekdayLong*

Back to top

yearday

```
pure elemental integer function yearday(self)
  class(datetime),intent(in) :: self
```

datetime-bound procedure. Returns integer day of the year (ordinal date). Equals to `1` for any January 1, `365` for a December 31 on a non-leap year, and `366` for a December 31 on a leap year.

Example usage

```
use datetime_module,only:datetime

type(datetime)  :: a

! Initialize:
a = datetime(2013,5,1)

print *, a % yearday() ! 121
```

See also

- *isocalendar*

Back to top

timedelta

Represents a duration of time, and a difference between two *datetime* objects. It is defined as:

```
type :: timedelta

  !! Class of objects that define difference between two datetime
  !! instances.

  private

  integer :: days         = 0 !! number of days
  integer :: hours        = 0 !! number of hours
  integer :: minutes      = 0 !! number of minutes
  integer :: seconds      = 0 !! number of seconds
  integer :: milliseconds = 0 !! number of milliseconds

  contains

  ! getter functions
  procedure,pass(self),public :: getDays
  procedure,pass(self),public :: getHours
  procedure,pass(self),public :: getMinutes
  procedure,pass(self),public :: getSeconds
  procedure,pass(self),public :: getMilliseconds

  ! public methods
  procedure,public :: total_seconds

  ! operator overloading procedures
  procedure,private :: timedelta_plus_timedelta
  procedure,private :: timedelta_minus_timedelta
  procedure,private :: unary_minus_timedelta
  procedure,private :: eq
  procedure,private :: neq
  procedure,private :: gt
  procedure,private :: ge
  procedure,private :: lt
  procedure,private :: le
  generic :: operator(+)  => timedelta_plus_timedelta
  generic :: operator(-)  => timedelta_minus_timedelta,&
                             unary_minus_timedelta
  generic :: operator(==) => eq
  generic :: operator(/=) => neq
  generic :: operator(>)  => gt
  generic :: operator(>=) => ge
  generic :: operator(<)  => lt
  generic :: operator(<=) => le

endtype timedelta
```

All arguments are optional and default to 0. Similarly to *datetime* objects, *timedelta* instances can be initialized using positional and/or keyword arguments. In addition, a *timedelta* object is a result of subtraction between two *datetime* objects.

Example usage

```
use datetime_module

type(datetime)  :: a,b
type(timedelta) :: c

! Initialize as default
c = timedelta()

! Positional arguments:
c = timedelta(0,1,15,0,0) ! 1 hour and 15 minutes

! Keyword arguments:
c = timedelta(days=1,hours=12) ! 1 day and 12 hours

! Difference between datetimes:
a = datetime(2013,5,12,22,0,0) ! 2013-05-12 22:00:00
b = datetime(2012,9,18,14,0,0) ! 2012-09-18 14:00:00

! Subtract to get timedelta:
c = a-b
```

Back to top

total_seconds

```
pure elemental real(kind=real_dp) function total_seconds(self)
  class(timedelta), intent(in) :: self
```

A *timedelta*-bound method that returns a number of seconds contained in the time interval defined by the *timedelta* instance. This method is equivalent to Python's *datetime.timedelta.total_seconds* function.

Return value `total_seconds` A total number of seconds (of type `real(kind=real_dp)`) contained in the *timedelta* instance.

Example usage

```
use datetime_module,only:timedelta

type(timedelta) :: td

td = timedelta(days=5,hours=12,minutes=15,seconds=7,milliseconds=123)

print *, td%total_seconds() ! 476107.12300000002
```

Back to top

clock

A generic clock object that contains start and stop times, tick increment and reset and tick methods. Most useful when needing to keep track of many *datetime* instances that change at different rates, for example, physical models with different time steps.

Definition:

```
type :: clock

  !! A clock object with a start, stop and current times, tick interval
  !! and tick methods.

  type(datetime) :: startTime
  type(datetime) :: stopTime
  type(datetime) :: currentTime

  type(timedelta) :: tickInterval

  logical :: alarm = .false.

  ! Clock status flags
  logical :: started = .false.
  logical :: stopped = .false.

  contains

  procedure :: reset
  procedure :: tick
```

```
endtype clock
```

clock instance must be initialized with some sane values of `clock%startTime`, `clock%stopTime` and `clock%tickIncrement` in order to be useful.

Example usage

```
use datetime_module

type(clock)    :: myClock
type(datetime) :: myTime

! Initialize myTime
myTime = myTime%now()

! Initialize myClock
! Starts from myTime, stops 1 hour later, 1 minute per tick
myClock = clock(startTime    = myTime,                   &
                stopTime     = myTime+timedelta(hours=1),&
                tickInterval = timedelta(minutes=1))

do

  call myClock % tick()

  ! Report current time after each tick
  print *, myClock % currentTime % isoformat(' ')

  ! If clock has reached stopTime, exit loop
  if(myClock % stopped)THEN
    exit
  endif

enddo
```

See also

- *datetime*
- *timedelta*

Back to top

reset

```
pure elemental subroutine reset(self)
  class(clock),intent(inout) :: self
```

Resets the clock to its start time.

Example usage

```
call myClock%reset() ! Resets myClock%currentTime to myClock%startTime
```

Back to top

tick

```
pure elemental subroutine tick(self)
  class(clock),intent(inout) :: self
```

Increments the `currentTime` of the clock instance by one `tickInterval`. Sets the `clock%stopped` flag to `.TRUE.` if `clock%currentTime` equals or exceeds `clock%stopTime`.

Example usage See *clock* for an example.

See also Back to top

tm_struct

Time object compatible with C/C++ *tm* struct. Available mainly for the purpose of calling *c_strftime* and *c_strptime* procedures.

```
type,bind(c) :: tm_struct

  !! A derived type provided for compatibility with C/C++ time struct.
  !! Allows for calling strftime and strptime procedures through the
  !! iso_c_binding.

  integer(kind=c_int) :: tm_sec   !! Seconds      [0-60] (1 leap second)
  integer(kind=c_int) :: tm_min   !! Minutes      [0-59]
  integer(kind=c_int) :: tm_hour  !! Hours        [0-23]
  integer(kind=c_int) :: tm_mday  !! Day          [1-31]
  integer(kind=c_int) :: tm_mon   !! Month        [0-11]
  integer(kind=c_int) :: tm_year  !! Year - 1900
  integer(kind=c_int) :: tm_wday  !! Day of week  [0-6]
  integer(kind=c_int) :: tm_yday  !! Days in year [0-365]
  integer(kind=c_int) :: tm_isdst !! DST          [-1/0/1]

endtype tm_struct
```

See also

- *datetime*
- *tm*

- *strftime*
- *strptime*

Back to top

Overloaded operators

datetime-fortran provides arithmetic and comparison operators for *datetime* and *timedelta* objects.

Arithmetic operators

Addition (`+`) and subtraction (`-`) operators are available for the following combination of derived type pairs:

- `datetime  + timedelta`, returns a `datetime` instance;
- `timedelta + datetime`, returns a `datetime` instance;
- `timedelta + timedelta`, returns a `timedelta` instance;
- `timedelta - timedelta`, returns a `timedelta` instance;
- `datetime  - datetime`, returns a `timedelta` instance;
- `-timedelta` (unary minus), returns a `timedelta` instance.

Note that `datetime  - datetime` operation accounts for timezone (`tz`) offsets in each of the `datetime` instances. The resulting `timedelta`thus includes the difference between timezones.

Comparison operators

datetime-fortran supports following binary comparison operators for *datetime* and *timedelta* objects: `==`, `/=`, `>`, `>=`, `<` and `<=`.

Since version 1.0.5, all comparison operators respect the timezone parameter of the datetime instances, so the operands are first adjusted to UTC time before making the comparison.

Back to top

Public procedures

c_strftime

```
function c_strftime(str,slen,format,tm) bind(c,name='strftime') result(rc)
  character(kind=c_char),dimension(*),intent(out) :: str
  integer(kind=c_int),value,          intent(in)  :: slen
  character(kind=c_char),dimension(*),intent(in)  :: format
  type(tm_struct),                    intent(in)  :: tm
  integer(kind=c_int)                             :: rc
```

An interface to a C/C++ standard library routine. Copies into `str` the content of format, expanding its format specifiers into the corresponding values that represent the time described in `tm`, with a limit of `slen` characters.

Note: This function was renamed from *strftime* to *c_strftime* in version 0.3.0 to avoid name conflict with *datetime*-bound method *strftime*. If working with *datetime* instances, use *datetime%strftime* instead.

Arguments `str` is the destination character string with the requested date and time.

`slen` is the maximum number of characters to be copied to `str`, including the terminating null-character, `char(0)`.

`format` is the character string containing any combination of regular characters and special format specifiers. These format specifiers are replaced by the function to the corresponding values to represent the time specified in `tm`. For more information on format specifiers see http://www.cplusplus.com/reference/ctime/strftime/.

`tm` is an instance of the type `tm_struct`, containing date and time values to be processed.

Return value If the resulting string fits in less than `slen` characters including the terminating null-character, the total number of characters copied to `str` (not including the terminating null-character) is returned. Otherwise, zero is returned and the contents of the array are indeterminate.

Example usage

```fortran
use datetime_module

type(datetime)     :: a
character(len=20) :: res
integer            :: rc

a = a % now()

rc = c_strftime(res,20,"%Y %B %d"//CHAR(0),a%tm())
```

See also

- *datetime%strftime*
- *c_strptime*
- *strptime*
- *tm*

- *tm_struct*

Back to top

c_strptime

```
function c_strptime(str,format,tm) bind(c,name='strptime') result(rc)
  character(kind=c_char),dimension(*),intent(in)  :: str
  character(kind=c_char),dimension(*),intent(in)  :: format
  type(tm_struct),                    intent(out) :: tm
  character(kind=c_char,len=1)                    :: rc
```

An interface to a C/C++ standard library routine. Converts the character string `str` to values which are stored in `tm`, using the format specified by `format`.

Note: This function was renamed from *strptime* to *c_strptime* in version 0.3.0 to avoid name conflicts with *strptime* which operates on *datetime* instances. If working with *datetime* instances, use *strptime* instead.

Arguments `str` is the character string containing date and time information.

`format` is the character string containing any combination of regular characters and special format specifiers, describing the date and time information in `str`.

`tm` is an instance of the type `tm_struct`, in which the date and time values will be filled upon successful completion of the *c_strptime* function.

Return value Upon successful completion, *c_strptime* returns the character following the last character parsed. Otherwise, a null character is returned.

Example usage Extracting time difference between two time strings using *c_strptime* and *tm2date*:

```
use datetime_module

type(datetime)  :: date1,date2
type(tm_struct) :: ctime
type(timedelta) :: timediff

! Return code for strptime
character(len=1) :: rc

! Example times in "YYYYMMDD hhmmss" format
character(len=15) :: str1 = "20130512 091519"
character(len=15) :: str2 = "20131116 120418"

! Get tm_struct instance from str1
rc = c_strptime(str1,"%Y%m%d %H%M%S"//char(0),ctime)
```

```
date1 = tm2date(ctime)

! Get tm_struct instance from str2
rc = c_strptime(str2,"%Y%m%d %H%M%S"//char(0),ctime)
date2 = tm2date(ctime)

timediff = date2-date1

print *, timediff
print *, timediff % total_seconds()
```

This example outputs the following:

```
        188             2           48            58          1000
   16253339.0000000
```

See also

- *strptime*
- *tm2date*

Back to top

date2num

```
pure elemental real(kind=real_dp) function date2num(d)
  type(datetime),intent(in) :: d
```

Returns the number of days since `0001-01-01 00:00:00 UTC`, given a *datetime* instance `d`.

This function is similar in what it returns to analogous functions in Python (*matplotlib.dates.date2num*) and MATLAB's *datenum*. Note that *matplotlib.dates.date2num* returns the number of days since `0001-01-01 00:00:00 UTC` plus `1` (for historical reasons), and MATLAB's *datenum* returns the number of days since `0000-01-01 00:00:00 UTC`. In *datetime-fortran*, we choose the reference time of `0001-01-01 00:00:00 UTC` as we consider it to be the least astonishing for the average user. Thus, MATLAB and Python users should be cautious when using *datetime-fortran*'s *date2num()* function.

Since version 1.0.5, date2num is timezone aware, i.e. the datetime instance is first converted to UTC before calculating the number of days.

date2num is the inverse function of num2date, so by definition, `a % utc() == num2date(date2num(a))` evaluates as `.true.` for any `datetime` instance `a`.

Arguments `d` A *datetime* instance.

Return value `date2num` A `REAL(KIND=real64)` number of days since `0001-01-01 00:00:00 UTC`.

Example usage

```
use datetime_module,only:datetime,date2num

type(datetime)  :: a

! Initialize:
a = datetime(2013,1,1,6)

print *, date2num(a) ! 734869.25000000000
```

See also

- *datetime*
- *num2date*

Back to top

datetimeRange

```
pure function datetimeRange(d0,d1,t)
  type(datetime), intent(in) :: d0
  type(datetime), intent(in) :: d1
  type(timedelta),intent(in) :: t
```

Given start and end *datetime* instances `d0` and `d1`, and time increment as *timedelta* instance `t`, returns an array of datetime instances. The number of elements is the number of whole time increments contained between datetimes `d0` and `d1`.

Arguments `d0` A *datetime* instance with start time. Will be the first element of the resulting array.

`d1` A *datetime* instance with end time. Will be the equal to or greater than the last element of the resulting array.

`t` A *timedelta* instance being the time increment for the resulting array.

Return value `datetimeRange` An array of *datetime* instances of length `floor((d1-d0)/t)+1`

Example usage

```
type(datetime)  :: a,b
type(timedelta) :: td
type(datetime),dimension(:),allocatable :: dtRange

a  = datetime(2014,5,1)
b  = datetime(2014,5,3)
td = timedelta(days=1)

dtRange = datetimeRange(a,b,td)

! Returns:
!
! dtRange = [datetime(2014,5,1),
!             datetime(2014,5,2),
!             datetime(2014,5,3)]

a  = datetime(2014,5,1)
b  = datetime(2014,5,3)
td = timedelta(hours=7)

dtRange = datetimeRange(a,b,td)

! Returns:
!
! dtRange = [datetime(2014,5,1,0),
!             datetime(2014,5,1,7),
!             datetime(2014,5,1,14),
!             datetime(2014,5,1,21),
!             datetime(2014,5,2, 4),
!             datetime(2014,5,2,11),
!             datetime(2014,5,2,18)]
```

See also

- *datetime*
- *timedelta*

Back to top

daysInMonth

```
pure elemental integer function daysInMonth(month,year)
  integer,intent(in) :: month
  integer,intent(in) :: year
```

Returns the number of days in month for a given month and year. This function is declared as `elemental`, so it can be called with scalar or n-dimensional array arguments.

Arguments `month` Integer number of month in year. Valid values are in the range [1-12].

`year` Integer year.

Return value Returns an integer number of days in requested month and year. Returns `0` if `month` is not in valid range.

Example usage

```
use datetime_module,only:daysInMonth

! January on leap year:
print *, daysInMonth(1,2012)   ! 31

! February on leap year:
print *, daysInMonth(2,2012)   ! 29

! February on non-leap year
print *, daysInMonth(2,2013)   ! 28
```

See also

- *daysInYear*

Back to top

daysInYear

```
pure elemental integer Function daysInYear(year)
  integer,intent(in) :: year
```

Given an integer `year`, returns an integer number of days in that year. Calls the *isLeapYear* function.

Arguments `year` An `integer` scalar or array containing the desired year number(s).

Return value `daysInYear` An `integer` scalar or array. Represents the number of days in `year`.

Example usage

```
use datetime_module,only:daysInYear

! Leap year:
print *, daysInYear(2012) ! 366
```

```
! Non-leap year:
print *, daysInYear(2013) ! 365
```

See also

- *daysInMonth*
- *isLeapYear*

Back to top

isLeapYear

```
pure elemental logical function isLeapYear(year)
  integer,intent(in) :: year
```

Returns a `logical` value indicating whether the reqested year is a leap year.

Arguments `year` An `integer` scalar or array representing year number.

Return value `isLeapYear` A `logical` scalar or array indicating whether a given year is leap year.

Example usage

```
use datetime_module,only:isLeapYear

! Leap year:
print *, isLeapYear(2012) ! .true.

! Non-leap year:
print *, isLeapYear(2013) ! .false.
```

See also

- *daysInYear*

Back to top

num2date

```
pure elemental type(datetime) function num2date(num)
  real(kind=real_dp),intent(in) :: num
```

Given the number of days since `0001-01-01 00:00:00 UTC`, returns a correspoding datetime instance.

This function is similar to analogous function in Python (*matplotlib.dates.num2date*).

num2date is the inverse function of date2num, so by definition, `a == num2date(date2num(a))` evaluates as `.true.` for any `datetime` instance `a`.

Similarly, `b == date2num(num2date(b))` evaluates as `.true.` for any variable `b` of type `real(kind=real64)`.

Arguments `num` Number of days since `0001-01-01 00:00:00 UTC`.

Return value `num2date` A *datetime* instance.

Example usage

```
use datetime_module,only:datetime,num2date

type(datetime)  :: a

a = num2date(734869.25d0) ! a becomes datetime(2013,1,1,6,0,0,0)
```

See also

- *date2num*
- *datetime*

Back to top

strptime

```
type(datetime) function strptime(str,format)
  character(len=*),intent(in) :: str
  character(len=*),intent(in) :: format
```

A wrapper function around *c_strptime*. Given a character string `str` with the format `format`, returns an appropriate *datetime* instance containing that time information. This function is analogous to Python's *datetime.datetime.strptime()* function. Available since version 0.3.0.

Arguments `str` is the character string containing date and time information.

`format` is the character string containing any combination of regular characters and special format specifiers, describing the date and time information in `str`.

Return value Upon successful completion, *strptime* returns the *datetime* instance corresponding to the time information contained in *str*.

Example usage Extracting time difference between two time strings using *strptime*:

```
use datetime_module

type(datetime)  :: date1,date2
type(timedelta) :: timediff

! Example times in "YYYYMMDD hhmmss" format
character(len=15) :: str1 = "20130512 091519"
character(len=15) :: str2 = "20131116 120418"

date1 = strptime(str1,"%Y%m%d %H%M%S")
date2 = strptime(str2,"%Y%m%d %H%M%S")

timediff = date2-date1

print *, timediff
print *, timediff%total_seconds()
```

This example outputs the following:

```
        188            2          48           58         1000
   16253339.0000000
```

This is the same example as in *c_strptime* but with fewer necessary steps.

See also

- *c_strptime*
- *tm2date*

Back to top

tm2date

```
pure elemental type(datetime) function tm2date(ctime)
  type(tm_struct),intent(in) :: ctime
```

Given a *tm_struct* instance, returns a corresponding *datetime* instance. Mostly useful for obtaining a *datetime* instance after a *tm_struct* is returned from *strptime*.

Arguments `ctime` A `tm_struct` instance.

Return value `tm2date` A `datetime` instance.

Example usage See example usage for *strptime*.

See also

- *datetime*
- *tm_struct*

- *strptime*

Back to top