

Time Manager

EcoSwarm software developed by:
Steve Jackson, Jackson Scientific Computing (jackson3@humboldt1.com)
Steve Railsback, Lang Railsback & Assoc. (LRA@Northcoast.com)

Overview and Conventions

The TimeManager class provides commonly used date and time functions. It is designed to support daily time step models and also models in which time is used in increments of hours or even down to seconds.

TimeManager makes extensive use of the operating system's Posix time functions, which use the "time_t" format. (Posix is a set of standards followed, more or less, by all major operating systems, so TimeManager's function should be independent of operating system.) A time_t value (which must be declared as type time_t) is an integer number of seconds since the start of January 1, 1970. Therefore, a time variable of type time_t can be incremented by an hour, for example, by adding 3600 to it.

One of the benefits of using this TimeManager instead of using the system time functions directly is that TimeManager circumvents the system's attempts to compensate for time zone and (especially) daylight savings time. The system functions usually used to convert a time_t back to date-time values (*localtime*, *strftime*) automatically attempt to adjust the time for daylight savings time, whereas simulation models (and the time series data used to drive them) typically do not consider daylight savings. Only the system function *gmtime* ignores daylight savings time, but *gmtime* converts the time to Greenwich Mean Time (GMT) instead of reporting local time.

TimeManager uses the system's *mktime* function to convert date-time values into time_t, telling the system to assume no daylight savings time; this creates time_t values for local standard time. Then, the difference between GMT and local standard time is added to the time_t value, converting it to GMT. Conversion from time_t to date and time values uses the system's *gmtime* function. TimeManager's internal use of GMT is not apparent to the user unless time_t values are saved and converted back to date and time outside TimeManager.

Users should be aware of two characteristics of TimeManager and the system time functions it uses. First, there is no error checking to ensure that the date or time used to create a time_t have realistic values. For example, the system function *mktime* will return a time_t value for November 55th, if requested, without generating an error. The only such error checking available is the **checkDateFormat**: method we wrote into the TimeManager (described below). Second, *mktime* uses a value of -1 as its error code in case it is unable to return a valid time_t. Therefore, TimeManager raises an error condition if *mktime* returns -1, even though -1 is the valid time_t for 23:59:59 of December 31, 1969.

The following formats and conventions are used for date and time values used by TimeManager.

- Dates are in the "MM/DD/YYYY" format (e.g., 12/31/2001). Leading zeros are acceptable but not necessary for the month and day (both 01/01/2001 and 1/1/2001 are valid).

- Dates include leap days.
- Day values (a month and day, with no associated year) are in the “MM/DD” format, with or without leading zeros.
- Hours are integers between 0 and 23. However, the function used to convert a date and hour to `time_t` format can also use input with hours between 1 and 24. Therefore, time series input in 1-24 hour format can be used, but all output created by TimeManager will be in 0-23 format. *Midnight is always treated as part of the following day.*
- Time values do not consider daylight savings time. Each day must have only hours 0-23, not extra or missing hours for when daylight savings time starts or stops.
- Minutes and seconds are integers between 0 and 59. TimeManager currently contains no code for handling minutes and seconds in 1-60 format, though it could easily be added.
- For `time_t` variables used to represent a date (year, month, and day) TimeManager uses a set of default values for the hour, minute, and second. For example, if the default hour, minute, and second are all set to 0, then the message `[getTimeTWithDate: 1/15/2001]` returns the `time_t` value for midnight at the start of 1/15/2001. These default values are initialized to represent noon: Hour = 12, minute = 0, second = 0. However, the user can use **`setDefaultHour:`** to specify new default values.

TimeManager has an instance variable `currentTime` which can be initialized to a `time_t` value and then advanced by a specified time step. This variable (accessed by the method **`getCurrentTime`**) can be used to keep track of time in a model that uses a fixed time step.

TimeManager objects can be created with or without a “controller” object. The controller object is treated as a password to ensure that the timeManager’s value of `currentTime` is not altered unintentionally. The method used to advance `currentTime` (**`stepTimeWithControllerObject`**) only works if provided with the same controller object specified when the timeManager was created.

Creating

TimeManager includes three alternative create methods. The first two are typically used to create a timeManager used by the model swarm to keep track of the model’s current time (as well as to furnish other time-related functions).

```
+ create: aZone
    setController: aController
    setTimeStep: (time_t) aTimeStep
    setCurrentTime: (time_t) aCurrentTime
```

Creates and returns a timeManager object. Sets the controller object (see **`stepTimeWithControllerObject`** below); typically the object creating the timeManager specifies itself as the controller object. The timeManager’s current time value is initialized to the value `aCurrentTime`.

```
+ create: aZone
    setController: aController
    setTimeStep: (time_t) aTimeStep
    setCurrentTimeWithDate: (char *) aFormattedDate
    withHour: (unsigned) anHour
    withMinute: (unsigned) aMinute
    withSecond: (unsigned) aSecond
```

Creates and returns a timeManager object. Sets the controller object (see **stepTimeWithControllerObject** below). Initializes the timeManager's current time value to the specified date (MM/DD/YYYY), hour, and second.

```
+ create: aZone
```

Creates and returns a timeManager object without setting any of its variables like controller, time step, current time, etc. This method is useful for creating a timeManager used only to make time-related calculations.

```
- createEnd
```

Used to complete creation.

Setting

```
- setDefaultHour: (int) anHour
    setDefaultMinute: (int) aMinute
    setDefaultSecond: (int) aSecond
```

Sets the default values used to define the time_t values for input specified only as a date. If the default hour, minute, and second are not set using this method, they retain their initialized values of 12, 0, 0 (noon).

```
- setController: aController
```

Provides an object that is used as a password to the **stepTimeWithControllerObject** method.

```
- setTimeStep: (time_t) aTimeStep
```

Provides the time step (number of seconds; internal variable timeStep) used to advance the variable currentTime using the **stepTimeWithControllerObject** method.

```
- setCurrentTime: (time_t) aTime
```

Re-sets the timeManager's variable currentTime to aTime. This variable can be used to keep track of simulation time in a model; see **setTimeStep**, **stepTimeWithControllerObject**, and **getCurrentTime**.

```
- setCurrentTimeWithDate: (char *) aFormattedDate
    withHour: (unsigned) anHour
    withMinute: (unsigned) aMinute
    withSecond: (unsigned) aSecond
```

Re-sets the timeManager's value of variable `currentTime`, using the specified date and time.

Using

```
- (time_t) stepTimeWithControllerObject: controllerObject
```

Increases the variable `currentTime` by the value of `timeStep`. Causes an error condition if (1) `controllerObject` is not the same object previously specified in a `setController` message, or (2) `timeStep` has not been set to a positive value. Returns the new `time_t` value for `currentTime`. This method is used to advance a model's simulation time.

```
- (time_t) getCurrentTimeT
```

Returns the current value of the variable `currentTime`.

```
- (time_t) getTimeTWithDate: (char *) aFormattedDate
```

Returns a `time_t` value for the date specified by the string `aFormattedDate` (MM/DD/YYYY). Uses the timeManager's default values of hour, minute, and second.

```
- (time_t) getTimeTWithDate: (char *) aFormattedDate
    withHour: (unsigned) anHour
    withMinute: (unsigned) aMinute
    withSecond: (unsigned) aSecond
```

Returns a `time_t` value for the input string `aFormattedDate` (MM/DD/YYYY) and hour, minute, and second.

```
- (char *) getDateWithTimeT: (time_t) aTime_t
```

Returns a new character string containing the formatted date (MM/DD/YYYY) for the specified `time_t` value. (This date string must be dropped if a memory leak is to be avoided.)

```
- getDateWithTimeT: (time_t) aTime_t
    modifyingMyString: (char *) myString
```

Copies the formatted date (MM/DD/YYYY) for the specified `time_t` value into `myString`, which must already exist. This method is useful for avoiding the need to create a new string every time the date is obtained.

- (int) **getYearWithTimeT:** (time_t) aTime_t

Returns the year (four-digit integer) for the specified time_t value.

- (int) **getMonthWithTimeT:** (time_t) aTime_t

Returns the month (two-digit integer, 1-12 for January - December) for the specified time_t value.

- (int) **getDayOfMonthWithTimeT:** (time_t) aTime_t

Returns the day of the month (integer between 1 and 31) for the specified time_t value.

- (int) **getHourWithTimeT:** (time_t) aTime_t

Returns the hour (integer between 0 and 23) for the specified time_t value.

- (int) **getMinuteWithTimeT:** (time_t) aTime_t

Returns the minute (integer between 0 and 59) for the specified time_t value.

- (int) **getSecondWithTimeT:** (time_t) aTime_t

Returns the second (integer between 0 and 59) for the specified time_t value.

- (int) **getJulianDayWithTimeT:** (time_t) aTime_t

Returns the Julian day (integer, number of days since December 31 of the preceding year) for the specified time_t value.

- **getJulianDayWithDay:** (char *) aFormattedDay

Returns the Julian day (integer, number of days since December 31 of the preceding year) for the specified input day string (MM/DD). Because the year is unspecified, this method assumes it is not a leap year.

- (time_t) **adjustTimeTToDefaultHMS:** (time_t) aTime_t

Returns a time_t that has the same date as the input aTime_t, but the hour, minute, and second are set to the timeManager's default values. This method is useful for converting a time_t that was created for any time within a day into a value that can be used in the timeManager's date methods.

- (time_t) **getTimeTForNextMMDD:** (char *) startMonthDay
 givenThisTimeT: (time_t) aTime_T

Finds the next date having its day (MM/DD) equal to startMonthDay, and occurring after the date specified by aTime_T.

- (BOOL) **checkDateFormat:** (char *) aDate

Checks to see if the string variable aDate is in the MM/DD/YYYY format used by TimeManager. Verifies that the month and day values have only two digits and fall within allowable ranges (month between 1 and 12; day between 1 and 31). Verifies that the year value has four digits. Returns NO if any of these checks are not successful.

- (time_t) **getSystemTime**

Returns the computer's system time (the clock value at the time the method was executed).

- (char *) **getSystemDateAndTime**

Returns a 35-character string with the computer's system time (the clock value at the time the method was executed), including the date and time.

- (BOOL) **getIsDSTWith:** (time_t) aTime

Returns YES if aTime represents a day that, according to the operating system's Posix time functions, falls within daylight savings time.

Dropping

- **drop**

Drops the timeManager and frees its memory.

Document History

First published version: 8/8/2001, SFRailsback.

Checked and revised: 6/30/2004, SFRailsback.