Community Atmosphere Model
National Center for Atmospheric Research, Boulder, CO

Time Manager Module: Requirements and Design

B. Eaton

20 November 2001

Online version:
http://www.cesm.ucar.edu/models/atm-cam/docs/time-manager/index.html

Contents

1 Overview

This document describes the requirements and design of the time manager module in the Community Atmosphere Model (CAM). The time manager is built on top of the utilities for time management available from the Earth System Modeling Framework (ESMF) [1] libraries.

The time manager's user interface is provided via namelist variables which may be set to control simulation properties such as the timestep size, start date, and stop date. Dates are specified using either a Gregorian or a 365-day calendar.

The time manager's API provides the methods that are used to control the model's time loop, and provides date and time information to model procedures or modules that may need it. The API also provides an alarm facility which is designed to be used by parameterizations that need to carry out certain actions at specified times during the simulation. The alarms are set up during the initialization process, and are queried each timestep for their on/off status.

2 Terminology

As a model advances in discrete timesteps the time manager keeps track of the date and time at each endpoint of the current timestep. We begin with this section on terminology to clarify our use of common words like "date" and "time."

date The term "date" is used to refer to an instant in time. It consists of year, month, day of month, and time of day components. The time of day is expressed in UTC. A date specification is incomplete without it's associated calendar.

time The term "time" is used in the sense of "simulation time" and expresses an elapsed time since a reference date. These time values follow the convention for time coordinates supported by the COARDS [2] and CF [3] metadata conventions.

time of day Time of day refers to the elapsed time since midnight of the current day. We express time of day in UTC unless explicitly stated otherwise. In common usage the term "time" refers to "time of day." In this document we will use "time of day" explicitly when that is what we mean.

start date The start date of a simulation is the date assigned to the initial conditions.

reference date The reference date of a simulation is an arbitrary date that corresponds to the origin of the time coordinate. Often this date corresponds to the start date. But it may be different because providing a reference date that is common to a set of simulations that may have different start dates allows them to use the same time coordinate.

current date A simulation advances by timesteps. At any point during a simulation the current date is taken to be the date at the end of the current timestep.

start time The start time of a simulation is the elapsed time from the reference date to the start date.

current time The current time of a simulation is the elapsed time from the reference date to the current date.

calendar day The day number in the calendar year. January 1 is calendar day 1. Calendar day may be expressed in a floating point format consisting of the integer day number plus the time of day (UTC) represented as a fractional day. For example assuming a Gregorian calendar:

Date	Calendar day
10 January 2000, 6Z	10.25
31 December 2000, 18Z	366.75

restart date The restart date of a simulation is the date of the data in the restart files from which the simulation is continuing.

3 Requirements

3.1 Date calculations

- Support Gregorian and "no-leap" calendars. A "no-leap" calendar is the same as a Gregorian calendar with no leap years.
- Represent dates and times to a precision of 1 second.
- Date and time representations must have a range of at least 200,000 years.

3.2 Simulation control

- The time manager will be initialized by specifying the calendar type, timestep size, start date and stop date. The user may optionally specify the reference date. By default the reference date equals the start date.
- The timestep size must be chosen so that there are an integral number of steps in a day.
- The stop date may optionally be specified by any of the following:
  - a number of timesteps from the start date;
  - a number of days from the start date;
  - a number of timesteps from the restart date;
  - a number of days from the restart date;
- The current date may be changed at any point during a simulation.
- Provide methods to query timestep size, start date, stop date, and reference date.

- Provide methods to query the following:
  - the start date, stop date, and reference date;
  - current timestep number and size;
  - current date, time, and calendar day;
  - previous date, i.e., date at the beginning of the current timestep;
  - perpetual date,
  - if the current timestep is the final one of the day or month;
  - if the current timestep is the first one of an initial or restart run;
  - if the current timestep is the last one of the run;
- The methods that return the current date and calendar day should provide for an optional offset to be specified. This provides easy access to the calendar calculations required to find date or calendar day values that are offset from the end-point of the current timestep.
- Provide an option to allow running the simulation in a "perpetual calendar" mode. Under this option the time and dates are always available both as the usual simulation time and date, and as a perpetual time and date. The simulation time and date are written to the output history files and log files to track the simulation progress with a monotonic time progression. The perpetual time and date are used to determine the sun position and interpolate boundary datasets. The perpetual day (i.e., the year, month, and day of month part of the date) may either be read from the initial file or may be set to user specified values. When running in aqua-planet mode the perpetual day is set to March 21 of year 0. The perpetual date and calendar day include a diurnal cycle.
  - A query method will be provided to determine if the time manager is using the perpetual calendar mode.
  - A query method will be provided that returns the current perpetual date which is the perpetual year, month, and day of month, plus the time of day component from the simulation's current date. This forces the time of day written to the history files to be consistent with the phase of the diurnal cycle that is used in the solar calculations.
  - The method that returns the current calendar day will return the calendar day corresponding to the current perpetual date when running in perpetual mode.
- The time manager must be able to "restart", i.e., it must be able to write its state to a binary restart file, and reset its state after reading the restart file. The only attribute of the time manager state that may be changed when initializing a restart run is the stop date.

3.3 Alarms

An alarm is used to signal that some event should take place during the current timestep. Alarms are initialized by specifying a set of times when they should be turned on. The time manager will turn an alarm on during the first timestep whose late end-point (i.e., the current time) equals or exceeds the current alarm time. When an alarm is turned on the alarm time is updated. Some specific requirements for alarm functionality are:

- Provide alarms that are turned on periodically and can be specified by a period and an offset.
- Provide alarms that are turned on when year and month boundaries are crossed.
- Provide method to query whether an alarm is on or off.
- Provide methods to query the current alarm time and date. The current alarm time is the time that the alarm is set to be turned on. If the alarm in currently on, then the current alarm time is the next time that the alarm will be turned on since the time manager updates the alarm time when it turns an alarm on.

4 Interface Design

The time manager is designed as a module which provides public data members for communicating with the user interface (i.e., Fortran namelist), and public methods which wrap the ESMF library methods. The reasons for designing "wrapper" methods rather than making direct use of the methods provided by the ESMF library's Fortran 90 interface are:

- to provide a simpler, higher level CAM specific interface;
- to hide the handling of the ESMF error returns;
- to hide the interprocess communication necessary when running with MPI;
- to provide methods for CAM specific file I/O to support restarts;
- to provide CAM specific simulation control options.

The ESMF library has an object oriented design. In Fortran this implies a viewpoint in which each variable declared as one of the derived types is thought of as an independent object. Thus, for example, different objects (variables) of the ESMF defined type for representing a date could be based on different calendars, and there could be multiple instances of the time manager. But in the context of a single atmospheric simulation there is only one calendar being used, and only one time coordinate. Hence, only one time manager is necessary. The CAM specific interface is then simplified, for example, by maintaining the calendar type and the time manager ID as private module data which does not have to be passed through the public method argument lists.

The ESMF library methods all provide an optional argument for error checking. The CAM specific interface does not pass this argument, but the methods act as wrappers that carry out checking of all error codes and invoke a CAM (or CCSM) specific error handler as appropriate.

The only CAM specific simulation control option currently implemented is to support a "perpetual calendar" mode. This mode is described above in the requirements section. The perpetual mode is currently used by the model's "aqua-planet" mode which is a testing configuration for the physical parameterizations.

4.1 User interface

The user interface is implemented via Fortran namelist variables. These variables are public data in the time manager module.

4.1.1 `calendar`

Type	character(len=*)
Default	'NO_LEAP'
Use	Calendar to use in date calculations. Supported calendars are 'NO_LEAP' (i.e., modern calendar, but without leap years) and 'GREGORIAN' (modern calendar).

4.1.2 `dtime`

Type	integer
Default	1200 s (Eulerian dycore), 1800 s (Finite-volume dycore), 3600 s (SLD dycore)
Use	Timestep size in seconds. `dtime` must evenly divide the number of seconds in a day.

4.1.3 `start_ymd`

Type	integer
Default	Read from the initial conditions input dataset
Use	Year, month and day of the simulation start date. The values are encoded in an integer as `year*10000 + month*100 + day`.

4.1.4 `start_tod`

Type	integer
Default	0 if `start_ymd` is specified; otherwise it's read from the initial conditions input dataset
Use	Time of day (UTC) of the simulation start date, in seconds.

4.1.5 `stop_ymd`

Type	integer
Default	none
Use	Year, month and day of the simulation stop date. The values are encoded in an integer as `year*10000 + month*100 + day`. If this value is specified it takes precedence over both `nestep` and `nelapse`.

4.1.6 `stop_tod`

Type	integer
Default	0
Use	Time of day (UTC) of the simulation stop date, in seconds.

4.1.7 `nestep`

Type	integer
Default	none
Use	If `nestep > 0` then `nestep` is the number of timesteps to advance the solution past the start date. If `nestep < 0` then the stop date is `-nestep` days past the start date. `nestep` is ignored if `stop_ymd` is set.

4.1.8 `nelapse`

Type	integer
Default	none
Use	If `nelapse > 0` then `nelapse` is the number of timesteps to advance the solution past the start date (on an initial run) or past the restart date (on a restart run). If `nelapse < 0` then the stop date is `-nelapse` days past the start or restart date (again depending on run type). `nelapse` is ignored if either `stop_ymd` or `nestep` is set.

4.1.9 `ref_ymd`

Type	integer
Default	from year, month, and day components of the start date
Use	Year, month and day of the time coordinate's reference date. The values are encoded in an integer as `year*10000 + month*100 + day`.

4.1.10 `ref_tod`

Type	integer
Default	from the time of day component of the start date
Use	Time of day (UTC) of the time coordinate's reference date, in seconds.

4.1.11 `perpetual_run`

Type	logical
Default	false
Use	Set to `.true.` to specify that the run will use a perpetual calendar. If `perpetual_ymd` is not set then read the perpetual day from the initial file.

4.1.12 perpetual_ymd

Type	integer
Default	none
Use	Perpetual date specified as `year*1000 + month*100 + day`. This date overrides the date from the initial file. If running in "aqua-planet" mode then `perpetual_ymd` is ignored and the perpetual date is set to `321`.

4.2 Application programmer interface

This section provides an overview of the design and functionality of the API. The following is a summary of the time manager's public methods in UML notation. In UML a method prototype has the syntax "`method-name (argument-list) :return-type`", where each argument in the comma separated list is expressed as "`intent arg-name:type`". The intent of an argument is one of the keywords `in`, `out`, or `inout`. Optional arguments are enclosed in brackets "[ ]". For methods which don't have a return value the "`:return-type`" is left off.

The API does not currently implement any alarm functionality because it is not yet fully functional in the ESMF library.

4.2.1 timemgr_preset

```
subroutine timemgr_preset()
```

`timemgr_preset` is used for run-time initialization of namelist variables. Most namelist variables are statically initialized in the time manager module. This method is currently used only for `dtime` whose default value depends on the dycore. This method is provided because `dtime` is used before the time manager is initialized (which happens after the header of the initial conditions file is read to obtain the default start date).

4.2.2 timemgr_init

```
subroutine timemgr_init()
```

`timemgr_init` initializes the time manager module for an initial run. Before `timemgr_init` is called it is assumed that the namelist has been read, and the date in the initial conditions file has been read.

4.2.3 timemgr_restart

```
subroutine timemgr_restart()
```

`timemgr_restart` initializes the time manager module for a restart run. Before `timemgr_restart` is called it is assumed that the namelist has been read, and the restart data has been read from the restart file by a call to `timemgr_read_restart`.

4.2.4 advance_timestep

```
subroutine advance_timestep()
```

`advance_timestep` advances the time manager by one timestep. This includes updating the date and time information at the beginning and end of the new timestep, and updating the alarms.

4.2.5 get_step_size

```
function get_step_size()
   integer :: get_step_size
```

`get_step_size` returns the current timestep size in seconds.

4.2.6 get_nstep

```
function get_nstep()
   integer :: get_nstep
```

`get_nstep` returns the current timestep number.

4.2.7 get_curr_date

```
subroutine get_curr_date(yr, mon, day, tod, offset)
   integer,           intent(out) :: yr, mon, day, tod
   integer, optional, intent(in)  :: offset
```

`get_curr_date` returns the components of the date corresponding to the end of the current timestep. When the optional `offset` argument is specified the current date is incremented by `offset` seconds (may be negative). The date components are: year (`yr`); month (`mon`); day of month (`day`); and time of day in seconds past 0Z (`tod`).

4.2.8 get_prev_date

```
subroutine get_prev_date(yr, mon, day, tod)
   integer, intent(out) :: yr, mon, day, tod
```

`get_prev_date` returns the components of the date corresponding to the beginning of the current timestep. The date components are: year (`yr`); month (`mon`); day of month (`day`); and time of day in seconds past 0Z (`tod`).

4.2.9 get_start_date

```
subroutine get_start_date(yr, mon, day, tod)
   integer, intent(out) :: yr, mon, day, tod
```

`get_start_date` returns the components of the date corresponding to the starting date for the simulation. The date components are: year (`yr`); month (`mon`); day of month (`day`); and time of day in seconds past 0Z (`tod`).

4.2.10 get_ref_date

```
subroutine get_ref_date(yr, mon, day, tod)
   integer, intent(out) :: yr, mon, day, tod
```

`get_ref_date` returns the components of the reference date part of the time coordinate. The date components are: year (`yr`); month (`mon`); day of month (`day`); and time of day in seconds past 0Z (`tod`).

4.2.11 get_perp_date

```
subroutine get_perp_date(yr, mon, day, tod)
   integer, intent(out) :: yr, mon, day, tod
```

`get_perp_date` returns the components of the perpetual date corresponding to the end of the current timestep. The date components are: year (`yr`); month (`mon`); day of month (`day`); and time of day in seconds past 0Z (`tod`).

4.2.12 get_curr_time

```
subroutine get_curr_time(days, seconds)
   integer, intent(out) :: days, seconds
```

`get_curr_time` returns the time at the end of the current timestep. `days` and `seconds` contain the components of the time interval in units of days and seconds respectively.

4.2.13 get_curr_calday

```
function get_curr_calday(offset)
   real(r8)                        :: get_curr_calday
   integer, optional, intent(in) :: offset
```

`get_curr_calday` returns the calendar day at the end of the current timestep. In perpetual mode the `get_curr_calday` returns the perpetual calendar day. When the optional `offset` argument is specified the current date (or perpetual date) is incremented by `offset` seconds (may be negative) before converting the date to a calendar day. The real kind, `r8`, specifies an 8-byte real value.

4.2.14 is_first_step

```
function is_first_step()
   logical :: is_first_step
```

`is_first_step` returns true during the initialization phase of an initial run. This phase lasts from the point at which the time manager is initialized until the first call to `advance_timestep`. In the CCM3 this phase was indicated by the conditional `if(nstep==0)...`

4.2.15 is_first_restart_step

```
function is_first_restart_step()
   logical :: is_first_restart_step
```

is_first_restart_step returns true during the initialization phase of a restart run. This phase lasts from the point at which the time manager is restart until the next call to advance_timestep. In the CCM3 this phase was indicated by the conditional if(nstep==nrstrt)...

4.2.16 is_last_step

```
function is_last_step()
   logical :: is_last_step
```

is_last_step returns true during the final timestep of a run.

4.2.17 is_end_curr_day

```
function is_end_curr_day()
   logical :: is_end_curr_day
```

is_end_curr_day returns true during the final timestep of each day. The final timestep of a day is the one whose ending date is equal to or later than 0Z of the next day.

4.2.18 is_end_curr_month

```
function is_end_curr_month()
   logical :: is_end_curr_month
```

is_end_curr_month returns true during the final timestep of each month. The final timestep of a month is the one whose ending date is equal to or later than 0Z of the first day of the next month.

4.2.19 timemgr_write_restart

```
subroutine timemgr_write_restart(ftn_unit)
   integer, intent(in) :: ftn_unit
```

timemgr_write_restart writes the state of the time manager to a binary file attached to Fortran unit ftn_unit. It is assumed that when running with MPI this method is only called from the master process.

4.2.20 `timemgr_read_restart`

```
subroutine timemgr_read_restart(ftn_unit)
   integer, intent(in) :: ftn_unit
```

> `timemgr_read_restart` reads the state of the time manager from a binary file attached to Fortran unit `ftn_unit`. It is assumed that when running with MPI this method is only called from the master process.

5 Module design notes

In discussions of valid ranges below we assume that the default native integer is signed and at least 4 bytes long.

5.1 Representation of dates

A date is represented by 2 integer values. One integer contains the calendar date (year, month and day of month), and the other contains the time of day (seconds past 0Z).

The year, month, and day of month components are packed in an integer using the expression `year*1000 + month*100 + day`. The range of valid dates is -2147480101 through 2147481231 which corresponds to a valid range of years from -214748 to 214748.

The time of day component of a date is represented as an integer number of seconds. Thus the precision of a date is 1 second.

The ESMF time manager defines a date type which may be initialized with the 2 integer values used by the CAM.

5.2 Representation of times

Time values are represented by 2 integer values. One integer contains the number of days and the other contains the partial day in seconds.

The valid range of times (which are assumed to be positive) is 0 through 2,147,483,647 days + 86399 seconds, or about 5.9 million years.

The precision of a time is 1 second.

The ESMF time manager defines a time type which may be initialized with the 2 integer values used by the CAM.

5.3 Counting timesteps

The underlying philosophy of the ESMF time manager design is that simulation control is based on the simulation date and the elapsed time from some reference date. There is no assumption that the timestep size is constant.

In the CCM3 the timestep size is fixed and the simulation control is based on counting timesteps. The timestep number and size plus a reference date are sufficient to compute the current date and time.

The ability to query the timestep number is for backwards compatibility with the CCM3 and allows a staged implementation of the time manager into the CAM. The alarm facilities in the time manager will eventually replace the logic that depends on the timestep number (which implicitly assumes a constant step size).

The timestep number in CAM2 is a native signed integer (usually 4 bytes) which implies that only 2^{31} steps may be counted. At the nominal timestep size of 1200 seconds this imposes a valid range of 81,000 years on the time manager.

5.4 Setting the stop date

The ESMF time manager determines if the current step is the last step of a simulation by testing whether the current date is equal to or later than the stop date. To support the options provided by positive values of the `nestep` and `nelapse` namelist variables, i.e., stopping the simulation a specified number of timesteps past either the start or current dates, the values must be converted to a corresponding stop date. This is done by incrementing the appropriate date by a time interval that is calculated using the current value of the timestep size multiplied by the specified number of timesteps. This implementation assumes a constant timestep size.

5.5 Status

- The ESMF time manager has support for resetting the current date at an arbitrary point in the simulation. This feature has not yet been incorporated into the CAM time manager.
- The ESMF time manager has not fully implemented the alarm functionality.

References

[1] Earth System Modeling Framework (ESMF).
http://www.esmf.ucar.edu/.

[2] "Conventions for the standardization of NetCDF files", Sponsored by the Cooperative Ocean/Atmosphere Research Data Service, a NOAA/university cooperative for the sharing and distribution of global atmospheric and oceanographic research data sets. May 1995.
http://ferret.wrc.noaa.gov/noaa_coop/coop_cdf_profile.html.

[3] NetCDF Climate and Forecast (CF) Metadata Conventions.
http://www.cgd.ucar.edu/cms/eaton/netcdf/CF-current.htm.