

Year Shuffler

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

Overview and Conventions

Often in long ecological simulations, the modeler is concerned that results can be partly an artifact of the sequence of annual habitat conditions. For example, the habitat input for a model might include some years with unusually good conditions and some years with unusually bad conditions; results of the model might be affected by how many good and bad years are simulated and the order in which they occur. One way to deal with this potential problem is to re-run the simulations with the years in different orders. YearShuffler is designed to support such analyses.

When created, a yearShuffler generates a randomized sequence of simulation years. Then, during a simulation, the **checkForNewYearAt**: method takes a date, checks to see whether a new simulation year starts on that date, and if so returns a date that has the next year from the randomized sequence.

Assumptions used in YearShuffler include:

- The only time period over which dates can be shuffled is a year: YearShuffler cannot, for example, be used to break a simulation into months and then randomize the sequence of months.
- Simulation years are defined by the simulation start date: if a simulation starts on October 1, then a different randomized year will be provided on each October 1.
- Date values are in time_t format; see EcoSwarm's TimeManager class for information on this format. This means that leap days are automatically accommodated.
- If the boolean parameter withReplacement is set to NO, then each year will be used only once. If withReplacement is set to YES, then the same year may be used more than once. (See below for more detail.)

When YearShuffler is used in the typical way, model runs with shuffled years require more data than runs without shuffled years, *if* the simulations are not exactly a whole number of years long. If, for example, a simulation runs from 1 January 1995 to 31 December, 2000 (exactly six years), then it will require input data for 1/1/1995 to 12/31/2000 no matter whether years are shuffled or not. However, if the simulation is set up to run from 1 August 1995 to 31 December 2000 and yearShuffler is to be used, then the input must cover the full period 1/1/1995 to 12/31/2000.

A yearShuffler creates its own random number generator (Swarm's default MT19937gen generator). If the same seed for this generator is used for multiple model runs, the runs

will all have exactly the same sequence of years; however, if the seed is modified among model runs the sequence of years will differ.

YearShuffler uses an instance of EcoSwarm's TimeManager class to handle its date and time calculations. Therefore, any model using a yearShuffler must also create a timeManager object. YearShuffler also uses the EcoSwarm class ZoneAllocMapper (which aids in allocating memory in such a way it can be dropped between replicate simulations), so any model using YearShuffler must include ZoneAllocMapper.

Creating

YearShuffler requires both the **createBegin** and **createEnd** methods to be used.

```
+ createBegin: aZone
    withStartTime: (time_t) aStartTime
    withEndTime: (time_t) anEndTime
    withReplacement: (BOOL) aReplaceFlag
    withRandGenSeed: (unsigned) aSeed
    withTimeManager: (id) aTimeManager
```

Creates and returns a yearShuffler object. Arguments aStartTime and anEndTime (in time_t format) define the start and end of the period over which years are shuffled. Normally, these will be the simulation start and end times. The argument aReplaceFlag is set to either zero (NO) or one (YES) to define whether the randomized sequence of years is created with or without replacement (see **createEnd**, below). The argument aSeed is used as the seed to the random number generator that generates the random sequence of years. The argument aTimeManager must be an instance of EcoSwarm's Time Manager class.

The only error checking is to make sure that aStartTime is less than anEndTime.

- (id) **createEnd**

Completes creation by generating a randomized sequence of years. The following steps are used.

First, the number of *complete and partial* simulation years between aStartTime and anEndTime is determined. For example:

- If aStartTime is the time_t value for 1 October 2000, and anEndTime is the time_t for 30 September 2002, then the number of years is 2;
- If aStartTime is 1 October 2000 and anEndTime is 29 September 2002, then the number of years is 2;
- If aStartTime is 1 October 2000 and anEndTime is 1 October 2002, then the number of years is 3; and
- If aStartTime is 1 January 2000 and anEndTime is 30 September 2000, then the number of years is 1.

Second, a list of years is created. This list starts with the year of `aStartTime`, and then subsequent years are added until the number of years on the list equals the number of complete simulation years determined in step 1. For example, if `aStartTime` is 1 January 2000 and `anEndTime` is 31 October 2005, then the list of simulation years is 2000, 2001, 2002, 2003, 2004, 2005.

Third, the list of years is shuffled into random order. If the `createBegin` argument `aReplaceFlag` is set to NO, then the randomized list of years is created simply by shuffling the original list using Swarm's ListShuffler class. In this "without replacement" case, the randomized list of years contains each of the original simulation years only once. If `aReplaceFlag` is set to YES, then the randomized list of years is created by randomly drawing its values from an even integer distribution that includes the same range of years as the original list of years. In this "with replacement" case, the randomized list can contain the same year more than once, and not all years between `aStartTime` and `anEndTime` may be in the list.

The list randomization uses a random number generator seeded with `aSeed`. Consequently, simulations using the same time period and the same value of `aSeed` will have years shuffled into exactly the same order. For simulations of the same time period to have differently shuffled years, then each simulation must use a different value of `aSeed`.

Using

- `(time_t) checkForNewYearAt: (time_t) aTimeT`

Accepts a `time_t` date argument and decides whether its value corresponds to the start of a new simulation year; if so, the `time_t` is modified to provide the new, randomized, year.

This method checks whether `aTimeT` has the same *month and day* as the argument `aStartTime` used in the `createBegin` method. It does NOT check whether `aTimeT` has the same hour, minute, or second as `aStartTime`. If `aTimeT` does not have the same month and day as `aStartTime`, then the method simply returns `aTimeT`.

However, if `aTimeT` does have the same month and day as `aStartTime`, then a new `time_t` value is created and returned. This new `time_t` has the same month, day, hour, minute, and second as `aStartTime`, but its year is the year pointed to by the current index to the randomized list of years that was created in `createEnd`. This new `time_t` is returned when the method finishes.

The `checkForNewYearAt:` method works by maintaining an index to the list of randomized years that was generated in `createEnd`. This index is set to zero at the end of `createEnd`. Each time `checkForNewYearAt:` detects the start of a new simulation year it uses the value of the index to determine which year in the randomized

list to use in generating a new time_t. Then, **checkForNewYearAt:** increments the index. This approach means:

- On the first day of a simulation, **checkForNewYearAt:** must be called. Because this first day will be the start of a simulation year, a time_t will be returned with a new, randomized year. If the first call to **checkForNewYearAt:** is not made on the first simulation day, then simulations will start with an un-shuffled date.
- **checkForNewYearAt:** must not be called more than once per simulation day. If it is, then the index to the randomized list of years will be incremented more than once on days that start a new simulation year. Incrementing the index more than once per new simulation year will eventually cause **checkForNewYearAt:** to read past the end of the list of randomized years.

It is common for a simulation using YearShuffler to include one more simulation year than in the unshuffled date sequence, when the same number of days are simulated. If aReplaceFlag is set to YES, the number of leap years in a simulation can change: leap years in unshuffled date sequence may not be included in the shuffled sequence. In this case, **checkForNewYearAt:** could attempt to read past the end of the list of randomized years. YearShuffler handles this situation by printing a warning statement and then returning dates from the first year of the randomized sequence. However, if **checkForNewYearAt:** attempts to read more than one year beyond the end of the list of shuffled years it raises an error condition and terminates execution.

- (id <List>) **getListOfRandomizedYears**

Returns the list containing the randomized sequence of years generated by **createEnd**. This list is a Swarm List object, which contains a list of objects. In this case, the objects on the list are pointers to integers containing the actual year values. For a model to find the first and second years in the randomized list, for example, the following code would be used.

```
int firstRandomYear;
int secondRandomYear;

listOfYearPointers = [myYearShuffler getListOfRandomizedYears];
firstRandomYear = (int *) [listOfYearPointers atOffset: 0];
secondRandomYear = (int *) [listOfYearPointers atOffset: 1];
```

Example Application

The following example code is taken from a trout model.

In the ModelSwarm's buildObjects method:

```
// Model parameter 'shuffleYears' determines whether yearShuffler
// is to be used.

if(shuffleYears == YES)
{
    //
    // Create the year shuffler and the data start and end times.
    //
    [self createYearShuffler];
    newYearTime = [yearShuffler checkForNewYearAt: modelTime];

    if (newYearTime != modelTime)
    {
        [timeManager setCurrentTime: newYearTime];
        modelTime = newYearTime;
    }
}
else
{
    modelTime = runStartTime;
    dataStartTime = runStartTime;
    dataEndTime = runEndTime + 86400;
}

*****

// This method actually creates the yearShuffler
// Parameters shuffleYearReplace, shuffleYearSeed are input
// to the Model Swarm.

- createYearShuffler
{
    startDay = [timeManager getDayOfMonthWithTimeT: runStartTime];
    startMonth = [timeManager getMonthWithTimeT: runStartTime];
    startYear = [timeManager getYearWithTimeT: runStartTime];

    endDay = [timeManager getDayOfMonthWithTimeT: runEndTime];
    endMonth = [timeManager getMonthWithTimeT: runEndTime];
    endYear = [timeManager getYearWithTimeT: runEndTime];

    yearShuffler = [YearShuffler createBegin: modelZone
                    withStartTime: runStartTime
                    withEndTime: runEndTime
                    withReplacement: shuffleYearReplace
                    withRandGenSeed: shuffleYearSeed
                    withTimeManager: timeManager];

    yearShuffler = [yearShuffler createEnd];

    if([[yearShuffler getListOfRandomizedYears] getCount] <= 1)
    {
        fprintf(stderr, "ERROR: TroutModelSwarm >>>> createYearShuffler >>>>
Cannot use year shuffler for simulations of one year or less\n");
        fflush(0);
        exit(1);
    }
}
```

```

    }

    //
    // Now calculate dataStartTime and dataEndTime
    // (the period over which we must have input data)
    {
        int numSimYears = [[yearShuffler getListOfRandomizedYears] getCount];
        int dataEndYear = [timeManager getYearWithTimeT: runStartTime] +
numSimYears;
        int dataEndMonth = startMonth;
        int dataEndDay = startDay;

        sprintf(dataEndDate, "%d/%d/%d", dataEndMonth, dataEndDay, dataEndYear);
        dataStartTime = runStartTime;
        dataEndTime = [timeManager getTimeTWithDate: dataEndDate
                                withHour: 12
                                withMinute: 0
                                withSecond: 0];

        dataEndTime = dataEndTime + 86400;

        fprintf(stdout, "TroutModelSwarm >>>> createYearShuffler >>>>
numSimYears %d\n", numSimYears);
        fprintf(stdout, "TroutModelSwarm >>>> createYearShuffler >>>> startYear
%d endYear %d\n", startYear, endYear);
        fflush(0);
    }

    return self;
}

*****

```

The ModelSwarm method that updates the model time, once per daily time step:

```

- updateModelTime
{
    time_t newYearTime = (time_t) 0;

    if(shuffleYears == NO)
    {
        if(initialDay == YES)
        {
            initialDay = NO;
        }
        else
        {
            modelTime = [timeManager stepTimeWithControllerObject: self];
        }
    }
    else // Do this if we're using yearShuffler
    {
        if(initialDay == YES)
        {
            initialDay = NO;
        }
        else
        {
            // Step the model time by one day
            modelTime = [timeManager stepTimeWithControllerObject: self];
        }
    }
}

```

```
// Check whether it is a new simulation year
newYearTime = [yearShuffler checkForNewYearAt: modelTime];

if(newYearTime != modelTime)
{
    [timeManager setCurrentTime: newYearTime];
    modelTime = newYearTime;
}
} // else initial day = NO
}

return self;
}
```

Document History

First draft: 1/20/2004, SFRailsback and SJackson.

Document revised, example updated: 7/1/04, SFRailsback.