

Parameter Manager

EcoSwarm Software Developed by Steve Jackson and Steve Railsback

Purpose

The Parameter Manager class was designed to facilitate models with agents of various subclasses that have much of their code at the superclass level. Creating one parameterManager object for each subclass provides a subclass-specific database of parameter values that can be accessed by code in the superclass. The Parameter Manager also performs testing to verify that parameters were initialized.

Our trout model is an example. The model has agents of superclass Trout and subclasses for each trout species. At the model level, we create an array of parameterManager objects, with the array indexed by the number of separate trout species being modeled. One parameter input file exists for each trout species and is used to load that species' parameterManager. Each trout object knows its species number (via its instance variable mySpeciesNdx). Therefore, in the superclass Trout code such as the following uses the species-specific parameter values:

```
fracMature = myLength/paramArray[mySpeciesNdx]->fishSpawnMinLength;
```

In this example statement, the fish calculates what fraction of sexual maturity it is by dividing its length by the parameter for the minimum length for spawning. The term

`paramArray[mySpeciesNdx]->fishSpawnMinLength` gets the value of the parameter `fishSpawnMinLength` for the appropriate trout species.

Compared to a more conventional approach of parameters being instance variables for each trout object, keeping parameters in a parameterManager object has these advantages: (1) Separate parameter sets for each species are handled easily; (2) Each fish does not have its own copy of the parameter values, reducing memory use, execution time, and error risk; (3) All parameters are declared in one isolated place (ParameterManager.h) instead of being mixed into the agent code; and (4) The Parameter Manager class provides a convenient place to check and test parameter values before simulations begin.

Verifying that parameters were initialized is an important function of the Parameter Manager. In the C language, uninitialized variables (such as parameters that were accidentally left out of an input file) are set to zero and execution continues without an error or warning being raised. This can easily cause important undetected errors in simulation results. The Parameter Manager uses Swarm's Probe facilities to verify that all parameters are initialized. During the createBegin phase, a complete probe map of all parameters is created. (This is done by creating a complete probe map, which includes probes to all variables in the Parameter Manager class and its superclass, then subtracting from the complete probe map the variables that are not model parameters.) Probes are then used to set all parameter values to null values. During the createEnd phase (after the model's parameter values are loaded), the probe map is used to verify that none of the parameters still have their null values.

The Parameter Manager currently supports variables of the following types: integer, float, double, string. It would require minor modification for use with parameters of other types.

Use

The ParameterManager is provided as example code that must be modified. The following steps are used to create a parameterManager object for used by one class of agents.

1. All parameters must be declared in the ParameterManager.h file. This is the only place they need to be declared.
2. All parameter names and values need to be included in an input file, following the Swarm Object Loader format.
3. The parameterManager is created (usually at the Model Swarm or agent superclass level) using its createBegin method.
4. Parameter values are loaded into the new parameterManager object using the Swarm Object Loader.
5. The parameterManager's createEnd method is executed. This method verifies that all parameters have been initialized.

Following is example code for creating and loading an array of parameterManager objects, with one such object per species of animal included in a model.

```
- _buildParamArray_ {  
  
    int numspecies;  
  
    // Create a nil array in the modelZone  
    paramArray = (id *) [ZoneAllocMapper allocBlockIn: modelZone ofSize:  
numberOfSpecies*sizeof(id)];  
  
    for(numspecies = 0; numspecies < numberOfSpecies; numspecies++)  
    {  
        paramArray[numspecies] = nil;  
    }  
  
    // Put a parameterManager object in the array for each species,  
    // using the Object Loader to get parameter values from the input files  
    for(numspecies = 0; numspecies < numberOfSpecies; numspecies++)  
    {  
        paramArray[numspecies] = [ParameterManager createBegin: modelZone];  
        [ObjectLoader load: paramArray[numspecies] fromFileName:  
            speciesParameterFile[numspecies]];  
        [paramArray[numspecies] setSpeciesIndex: numspecies];  
        paramArray[numspecies] = [paramArray[numspecies] createEnd];  
    }  
  
    return self;  
}
```

Parameter values for an agent are then obtained as:

```
paramArray [speciesNdx] ->parameterName
```

where `speciesNdx` is the agent's species number and `parameterName` is the name of the desired parameter.

The Parameter Manager also includes a method `printSelf` that prints all parameter values to a file. This method is useful for testing and debugging.