Probing and Displaying the Contents of a SwarmObject

Contents:

1. Probe Overview

2. The Framework

3. Precision setting for floats with Probes

4. SwarmObject Support for Probing

5. ProbeDisplay

6. ProbeDisplayManager

Overview

The Framework

Probes, ProbeMaps and ProbeDisplays allow the user to dynamically interact with the objects in their simulation. By *interaction* we mean reading/setting instance variables as well as dynamically generating method calls. The main point being that these interactions are not hardwired into the program code, but occur due to user-generated requests, mainly through the provided GUI.

The key to this capability is the Probe: in general a probe takes an object and either extracts the value of a specific variable, or calls a specific method. For this purpose we provide two subclasses: VarProbe and MessageProbe.

There are two main uses for probes: they can be fed into data-collection objects and serve as interfaces to the objects about which data is being collected (thus keeping the data-collection objects as general as possible) - the Averager class, for example, directly subclasses MessageProbe. Or, they can be used in order to generate a GUI to the individual objects in the simulation (the more common usage).

In order to generate a graphical version of a given probe, the programmer must place it inside a ProbeDisplay, which will automagically generate a window with the appropriate interface. Since, more often than not, the programmer will want to generate windows with more than one variable, we have designed the ProbeDisplay to deal with ProbeMaps rather than individual probes. Thus, by generating a ProbeMap containing exactly the right probes, a user is able to customise the window generated by the ProbeDisplay.

In order to facilitate the creation of all these different objects, the Swarm kernel provides some alternative methods for Probe/Map/Display generation:

- Direct Generation - which leaves the user in charge of all the details.
- Object Generation (Probes/ProbeMaps) - where we use method calls (defined in SwarmObject) on a given target object to generate the Probe/ProbeMap for that object. This allows the object to be aware that it is being probed and to control directly the contents of the Probe/ProbeMap being requested...
- Library Generation (Probes/ProbeMaps) - where the programmer "checks out" a unique, shared copy of a Probe/ProbeMap from the probeLibrary object (of class ProbeLibrary) provided by the kernel. By shared we mean that a similar request made at a different point in the code, will return a reference to the very same probe instance.
  Note that, by default, Object Generation is equivalent to Library Generation since the default behaviour of SwarmObjects, when requested to create a ProbeMap, is to return a reference to the probeLibrary's unique copy!!!
- ProbeDisplayManager Generation (ProbeDisplays) - where the programmer generates a ProbeDisplay directly, by requesting it from the probeDisplayManager object (of class ProbeDisplay) provided by the kernel (in graphics mode). The probeDisplayManager will create the ProbeDisplay based on a ProbeMap given to it by the probed object. Note: since the probeDisplayManager is only created in graphics mode, it is documented in the simtools area of the documentation. Here we emphasise simply that in order to generate a ProbeDisplay onto an object the programmer need only write the following line of code:

```
  [...]

  probeMap = [EmptyProbeMap createBegin: aZone];
  [probeMap setProbedClass: [self class]];
  probeMap = [probeMap createEnd];
  // set the display defaults
  [probeLibrary setDisplayPrecision: 3];
  // typically saved precision would be higher than displayed precision
  // for statistical and data analysis purposes
  [probeLibrary setSavedPrecision: 10];

  // Add in a bunch of variables, one per simulation parameters
  [probeMap addProbe: [probeLibrary getProbeForVariable: "displayFrequency"
    inClass: [self class]]];
  [...]

  // Now install our custom probeMap into the probeLibrary.
  [probeLibrary setProbeMap: probeMap For: [self class]];
  return obj;
}
```

Setting precision for individual probes

The formatting for an individual probe can be set directly, using a sprintf-style formatting string. Typically, `customProbeMaps` are created in the `+ createBegin` factory method for a Swarm or a SwarmObject. To set the formatting for a floating point probe, the method from `VarProbe`

`(objectbase/VarProbe.html)`
is used:

```
-setFormatFloat:  (const char *)floatFormat
      is applied to the instance of the VarProbe "checked-out" of the global probe
      library instance.  The sprintf-formatting string can "over-ride" the "%g"
      format set by the global precision (as above) (Typically "%g" chooses between
      the "%f" and "%e", depending on the size of the exponent - which is implementation-depe
      - this method allows you to explicitly set the type of display).
```

In the following example, it is desired that the number of significant figures for the floating point variable `randomMoveProbability` is three (3). Currently (Swarm 1.0.5) this is only works for `VarProbes` and not `MessageProbes`, as yet.

Example Here's how it's done in `HeatbugModelSwarm.m`:

```
+ createBegin: aZone
{
  HeatbugModelSwarm * obj;
  id <ProbeMap> probeMap;
```

```
        If you want to put a specific title on the ProbeDisplay window associated
        with your object, simply override this method.  By default it returns the
        class name of the object.  This is why the default ProbeDisplay window title
        is always the class name of the probed object.


Note:  SwarmObjects do not actually have an instance variable called "instanceName"
- so if you want to store a name for your object, you will need to provide your own
storage at the subclass level and rewrite getInstanceName to refer to this variable.


  - getProbeForVariable:  (const char *)aVariable


  - getProbeForMessage:  (const char *)aMessage


  - getProbeMap


  - getCompleteProbeMap
```

Probe support resident in the *simtools* library:

ProbeDisplay

The `ProbeDisplay` class is meant to serve a dual role, represented by the two example windows on this page:

slot and release the mouse button).

- Also, if an instance variable/argument slot is defined to hold an object, then that object can be inspected by clicking the entry for that variable/argument slot with the third mouse button(a ProbeDisplay for that object will be generated).

Available only on the standard ProbeDisplay:

- Note that the sunken label at the top of the ProbeDisplay is also active. By clicking on it with the first mouse button you get a drag&drop'able representation of *self*. By clicking on it with the third mouse button you get a CompleteProbeDispl to *self*.

Available only on the ProbeDisplay:

- The green "superclass" button can be used to display the succesive superclasses of the object being probed.
- The red "hide" button can be used to hide classes which are irrelevant thus reducing clutter.
- The hide button on the lowest class in the hierarchy has a special meaning since clicking on it dismisses the entire ProbeDisplay.

Interface definition

(simtools/ProbeDisplay.html)
from simtools
(simtools.html)
library.

ProbeDisplayManager

The normal Swarm simulation will probably only ever contain one instance of this class, namely the probeDisplay-Manager. This object is used for automatic generation of ProbeDisplays: given an object it will attempt to get a ProbeMap from the object. If the object does not respond to the method - getProbeMap (i.e. it does not inherit from SwarmObject), the probeDisplayManager will query the probeLibrary for a ProbeMap. If there is not a *ProbeMap* installed in the probeLibrary for the class of that object, it will creat a *DefaultProbeMap*. It will then create a ProbeDisplay to the target object, and from then on, until the Probe Display is removed it will update the ProbeDisplay when receiving an update message.

Interface definition
(simtools/ProbeDisplayManager.html)
from simtools
(simtools.html)
library.