# Some notes on programming objects in The Sims™

Kenneth D. Forbus
Qualitative Reasoning Group
Northwestern University

Will Wright
Maxis/Electronic Arts

**Acknowledgements:** Maxis/Electronic Arts is in no way responsible for this material, nor do they distribute or support the Edith object editor.

# 1 Introduction

The purpose of this document is to help you get started creating objects in The Sims. It consists of three parts:

1. An under-the-hood look at The Sims. Understanding the basics of how this simulated environment is structured is essential for understanding how to create the routines and marshal the resources necessary to create new objects.
2. A guide to the Edith object editor and debugger. The interpreted nature of the simulation environment greatly simplifies the debugging process, and is what makes it possible to drastically extend the simulation by adding new objects. Edith lets you watch what is going on inside objects and people while the simulation is running, as well as providing facilities for creating and editing routines using the visual programming language of the underlying virtual machine.
3. Some examples of object creation. We start with creating a Joy Booth. A more complex exercise, a Mood Adjuster, is described. Some yet more complex exercises are suggested.

# 2 The Sims: A look under the hood

The Sims' world is created on top of Edith, which provides a virtual machine plus an development environment for that virtual machine. Shipping versions of The Sims have the virtual machine but not the development environment.

A simulation in Edith consists of a set of objects. The simulation is evolved by running each of these objects in turn. Each object has its own (Edith-level) thread and local variables. The Edith VM uses cooperative (i.e., non-preemptive) multitasking, so it is important to include explicit global checks in your code to avoid blocking.

Every object has a set of local data and a set of behaviors. The local data provides the parameters of an object, including pointers (indexes really) to other objects it is related to. The set of behaviors consist of a procedure that implements it (more on this below), a procedure that checks to see whether or not it is possible, and a set of *advertisements* that describe its properties in terms of what need(s) of a Sim it will satisfy. Sims (not under direct player control) choose what to do by selecting, from all of the possible behaviors in all of the objects, the behavior that maximizes their current happiness. Once they choose a behavior, the procedure for that behavior (which is part of the object) is then run in the thread of the Sim itself, so that it has access to that Sim's parameters in addition to those of its defining environment (the object the behavior is from). All interactions between objects occur in this way. Sims themselves are just a somewhat more elaborate object.

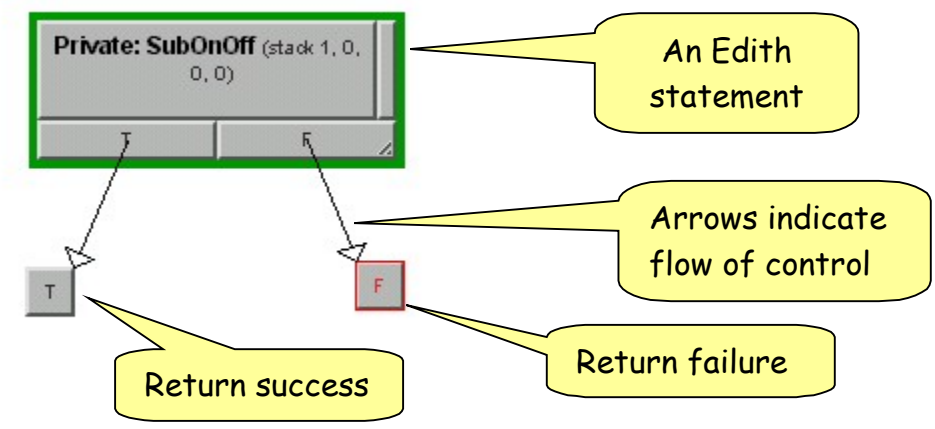
Sometimes there are a number of intermediate objects to implement behaviors. For instance, Social Interaction is an object that is created and used when two Sims interact.

Each Sim contains behavior calls for each of the possible social interactions (flirt, kiss, etc). When SimSam decides to run the kiss behavior (which is in SimMary) an invisible social interaction object is created. The execution of SimSam's thread is then passed into this object (as is SimMary's if she's not busy). (Access to the parameters of the chosen Sim is provided through a pointer in the Social Interaction object.). Eating is another example of a behavior that involves a large number of intermediate objects, basically one per step. Notice that this means objects can be dynamically created and destroyed during the execution of a behavior.

Although the object updates happen in a single (underlying processor) thread, animations and sounds are executed in another thread, asynchronously with Edith-level operations. That is what provides the illusion of simultaneous activity in the simulated world. This is also why there is a fair amount of setup that must be done when entering a routine that uses animation, and careful checking on exit. There are a number of subtleties with animations. For instance, when a Sim is taking a shower, it doesn't actually enter the shower. Instead, it is added to a "routing slot" of the shower, which offsets the location of the animation so that it seems that the Sim is in the shower. (Try executing another behavior when the Sim is in the shower, by editing the Shower Core procedure, and you'll see teleportation in action. Implementing a realistic simulated world is far from easy!)

Behaviors are implemented in terms of procedures, called *trees* because of the visual programming language that Edith uses. The intent of the visual programming language is to make it easier for content developers to create objects, and perhaps someday support end-user programming.[1]

Statements in this visual language are represented by boxes. Here is a very simple behavior, the behavior for turning on the Lava Lamp:




[1] Imagine an introductory programming course where your examples were in terms of Sims objects and behaviors, rather than writing simple address books or matrix multiplication routines. This is still a great idea, but it will take a lot of good curriculum design work to make it happen, in addition to more bulletproofing.

The entry point of any procedure is highlighted in green. Holding the control-key and left-clicking on any procedure will make it the entry point for that tree. A statement is either a primitive or a procedure call. The **Private:** indicates that the procedure is local to the object, **Global:** indicates that the procedure is global. The small T/F boxes indicate places where control is returned from the procedure, with success or failure respectively. Arrows indicate flow of control. Notice that the statement in this procedure has two tabs, one T and one F, each representing a branch to take depending on whether or not the statement succeeded or not. This is how conditionals are implemented in this language. If a statement doesn't have a conditional outcome, then only a single tab, with the label T, appears at the bottom. Iteration is done via loops in the branching structure, with special primitives (e.g., **Set To Next**) acting as generators.

Primitive expressions enable the Edith programmer to test, set, and mutate parameters, to run animations and play sounds, and do various other things. Integer id's are used to refer to objects.

# 3 The Edith object editing and debugging environment

The visible portion of Edith is the object editing and debugging environment. This section provides enough information to get you started, but for the rest, you're on your own.

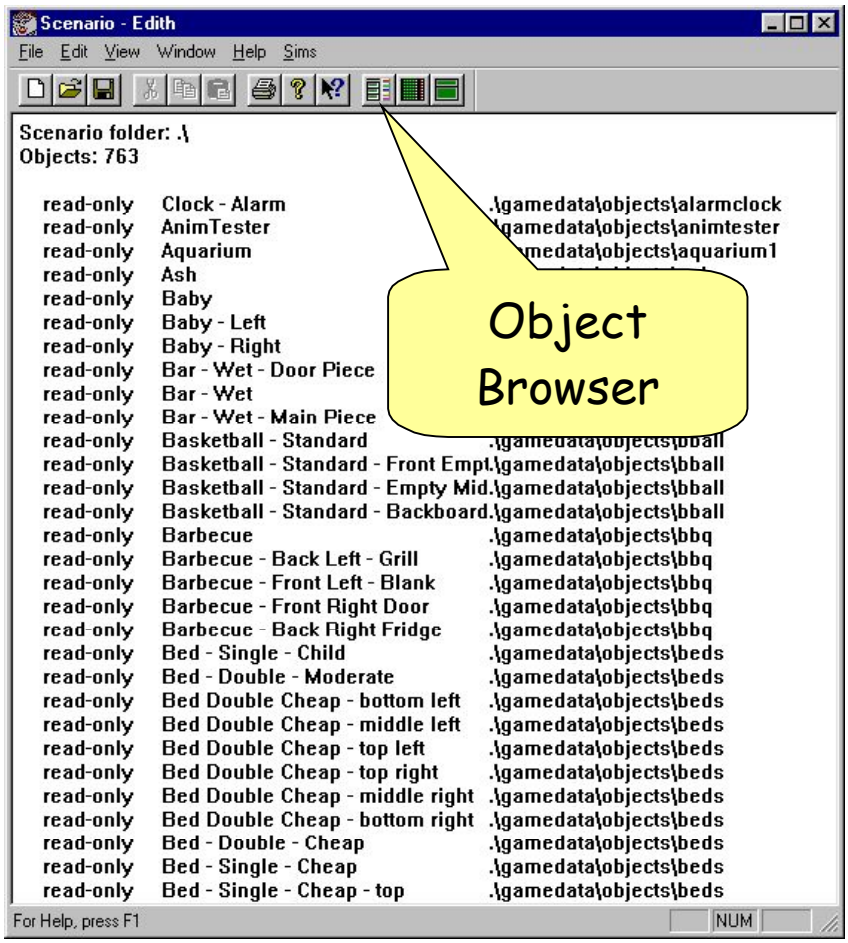
## *3.1 Installing Edith*

Copy `SimE.exe` into the directory containing your legally obtained installation of The Sims. Please note: This version of Edith is not the latest version, and will only work with The Sims, not any of the expansion packs. Also, add the objects that are on the class CD to the GameData\Objects directory, and change their properties so that they are no longer read-only. Otherwise, you will not be able to edit them.

## *3.2 Starting up Edith*

It is important to note that Edith requires your display color depth to be set to 16 bits. Moreover, you can only run Edith in windowed (as opposed to full screen) mode. To do this, set up a shortcut which executes `SimE.exe`, but with `-w` as a command line parameter. When you execute your shortcut, you'll see The Sims starting almost as usual, albeit in a window rather than taking up your entire screen. Choose a family to work with, and then type "`e`" to bring up the Edith window.

When Edith starts up, you'll see a scenario editor window. This lists information about the objects involved in the scenario you are currently running, i.e., the house and family you have chosen to look in on. Here's an example of what you might see:




Most of the buttons are standard, but the really useful stuff is under the Sims menu and the third button from the right, which starts the Object Browser.

## *3.3 The Object Browser*

The object browser provides tools for programming objects. Click on an object to select it, then choose what you want to do with it. The key buttons to know about are:

- Edit Tree Table brings up the dialog for editing
- Edit Behavior brings up the dialog for editing routines associated with an object.

## *3.4 Edit Tree Table Dialog*

You'll notice that there is no cancel button on this dialog. Any changes you make will not be actually recorded until you click on Save. You have been warned…

The Interactions panel describes the interactions that an object makes available. The index indicates what order something shows up in the menu. The Check Tree is a piece of code that indicates whether or not that interaction should be available, given the state of the object. If this code exits false then that behavior will not appear on the pie menu and also will not be available for autonomous selection by the Sims. The Action tree is the action that is executed when that interaction is executed. The set of interactions can be edited with the buttons in this panel. Most of them are obvious (New adds a new interaction, Delete removes the selected interaction, Set/Clear Check/Action buttons change those aspects of the interactions. The combo boxes are used to select actions, pulling them from the pool of procedures (aka trees). Private trees are stored with an object. Global trees are available to every object. Semi-global procedures appear to be unused.

The Interaction Data panel sets up how an interaction of an object affects people. Recall that objects provide interactions, and these interactions are advertised to the Sims. Here's what the parts do:

- Menu name is the name of that interaction displayed on the menu that players see when they click on the object.
- Attenuation sets how quickly the advertisement fades. A setting of none makes the strength of the advertisement independent of distance. In addition to the preset values (high, moderate, low), one can set a custom value.
- Flags indicate to whom this interaction advertises. The "Available to" flags are obvious. Allow Consecutive means that this interaction can be queued up.
- Joining is something you should just ignore.
- Motives indicate how the appeal of the advertisement is modulated by the properties of the Sim. Recall that the Sims are constantly seeing interactions that will satisfy their needs (energy, comfort, hunger, hygiene, bladder, room, social, fun, mood). The motive settings determine how strong the advertisement will be along different dimensions. For instance, a toilet advertises to Bladder, in a range from 0 to 70 (set by the Min and Max boxes). If there is no variation by personality, the Max is used. If it does vary by personality, then the motive is scaled linearly according to the number of points for that aspect of the personality. For instance, the aquarium's Feed Fish interaction provides between 1 and 11 on Fun, depending on the Playful personality component. You'll notice that the personality list actually includes polar opposites for each trait (i.e., nice/grouchy, active/lazy, etc.) This is an intuitive way of handling a change of sign, i.e., if the personality variation is chosen to be Serious, the more playful points the Sim has, the closer that dimension will be to the minimum.

## *3.5 Tweaking your Sims*

A good way to learn how a simulator works is to tweak its parameters and see how its behavior changes. The Sims menu in the scenario window provides several ways to do this:

- The Simulation Constants menu enables one to reset basic properties of the Sims, basically the rates at which various parameters change over time.
- The Simulation Globals menu enables changing fundamental parameters of the simulation, such as the person selected or whether or not free will is enabled, and properties of the household, such as the money available.

### *3.6 Edith tips, tricks, and traps*

- Back up your GameData directory frequently when programming with Edith. Mysterious crashes can happen.
- Although you can do it, it is probably very unwise to edit global procedures unless you are extremely sure you know what you're doing. You can break things very severely if you do. `Global:Wait for Notify` is one example. Just back out, and let it do its job.
- Tweaking personality parameters and states of objects is relatively easy to do. Working with animations is really hard. The animations were made with 3D Studio Max plus a number of plug-ins that are EA proprietary, and they simply aren't available. The animations can include events, which is one reason why they are so tricky.
- When editing procedures, make sure that you've terminated every node with either a pointer to another node or an exit (T/F box). Otherwise very bad things can happen. This is the most common mistake when working in Edith.
- Game tuning with new objects is a lot easier if you can gather data without keeping your eyes glued to the screen. You can create a log file by invoking the cheat window (Control-shift-C) and typing `sim_log begin`. Lots of data is dumped into a .txt file in your Sims directory, until you reenter the cheat window and type `sim_log end`. The data is in a format that can be read into a spreadsheet for further analysis.
- You'll find other handy abilities in the cheat window as well. For instance, you can set the hour so that time-based operations can be tested. `Help` gives you a listing of commands.
- Right-clicking on a box sets a breakpoint.
- If you find tons of small text label boxes showing up when you look at a lot, one per object, you probably have redundant objects in your GameData\Objects directory. Move the extra .iff files somewhere else and this problem will go away.

# 4 Designing new objects

Consider balance when creating your objects. Life is full of tradeoffs, and navigating the constraints that they impose is part of the fun. In the game, objects that confer some benefit should also have some cost. The most interesting costs are those which are

indirect – unintended consequences of something that looked like a good idea. We'll see that in our first object…


## *4.1 The Joy Booth*

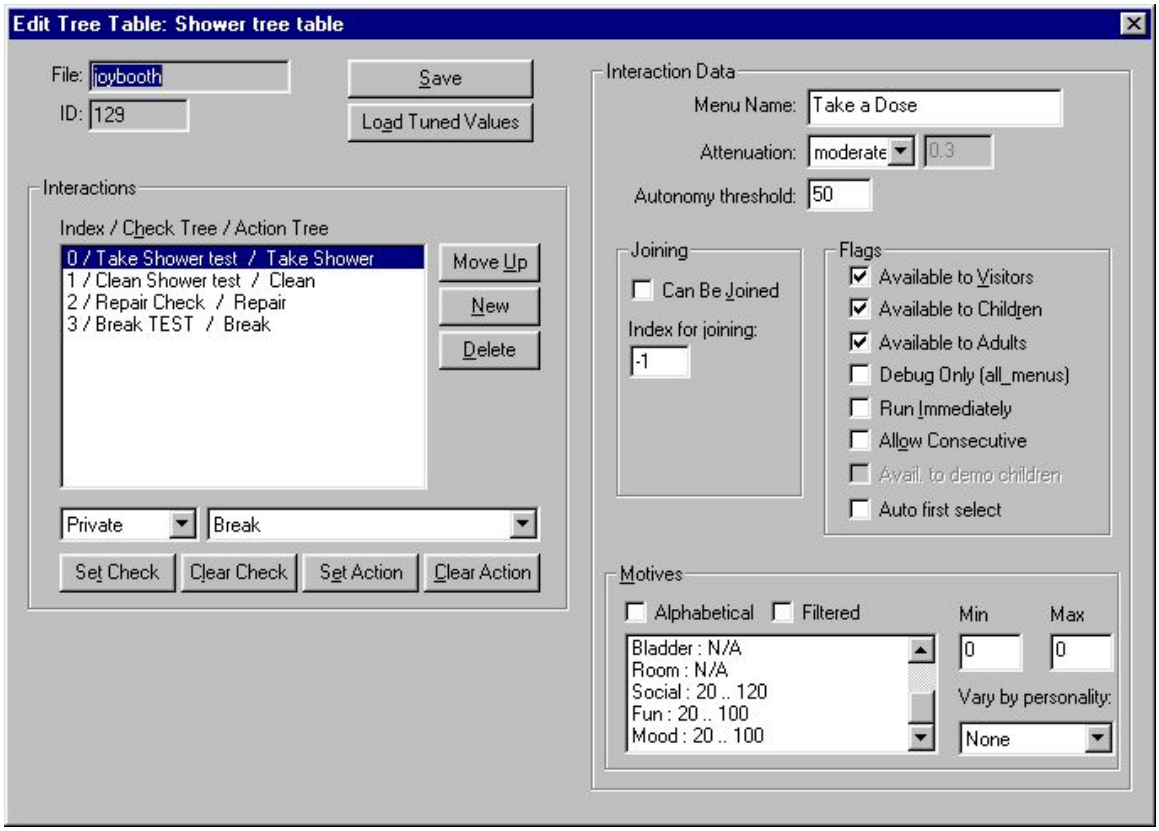
The Joy Booth is based on the Joy Booths in the classic Infocom game, "A Mind Forever Voyaging" and on the Orgasmatron from Woody Allen's classic movie "Sleeper". The Joy Booth catalog description is shown above. When you instruct a Sim to use the Joy Booth, it looks something like this:


On the right is a picture of the Joy Booth in operation. As you watch a Sim using it, you'll notice that their motions are like taking a shower, albeit with their clothes on. Joy Booths quickly become popular household items, fun for the whole family and the neighbors:

The down side is that it is so popular that the family and the neighbors use it so much that they socialize much less, thus weakening relationships, and even lose sleep to get another dose. We've created a kind of SimHeroin, if you will, something that the Sims find altogether too addicting.

How was the Joy Booth built? It was created in two phases. First, the Transmogrifier was used to create the Joy Booth visuals and change the catalog entry and price. The Transmogrifier documentation is excellent, so you can read that to find out how to use it. The only caveat I would add is to be sure to give your new object a unique ID – otherwise it will not show up in the catalog. The Joy Booth started as a transmogrified shower, with new textures. The shower was the appropriate starting point because it is one of the few objects that Sims actually enter. (Recall that doing new animations outside Maxis is currently impossible, since they require proprietary tools.) The second phase is to use the Edith editor to reprogram the routines of the Shower that are copied over during the transmogrification to make the new object behave like a Joy Booth.

How should a Joy Booth work? For any object in the Sims, we have to consider what behaviors it should have and how these behaviors should advertise themselves. A Joy Booth is described as being useful for saving time in avoiding relationship maintenance. This suggests that taking a dose from the Joy Booth should provide Social and Fun payoffs. As a bonus, we'll provide a Mood payoff as well. Here are the values for the Joy Booth:



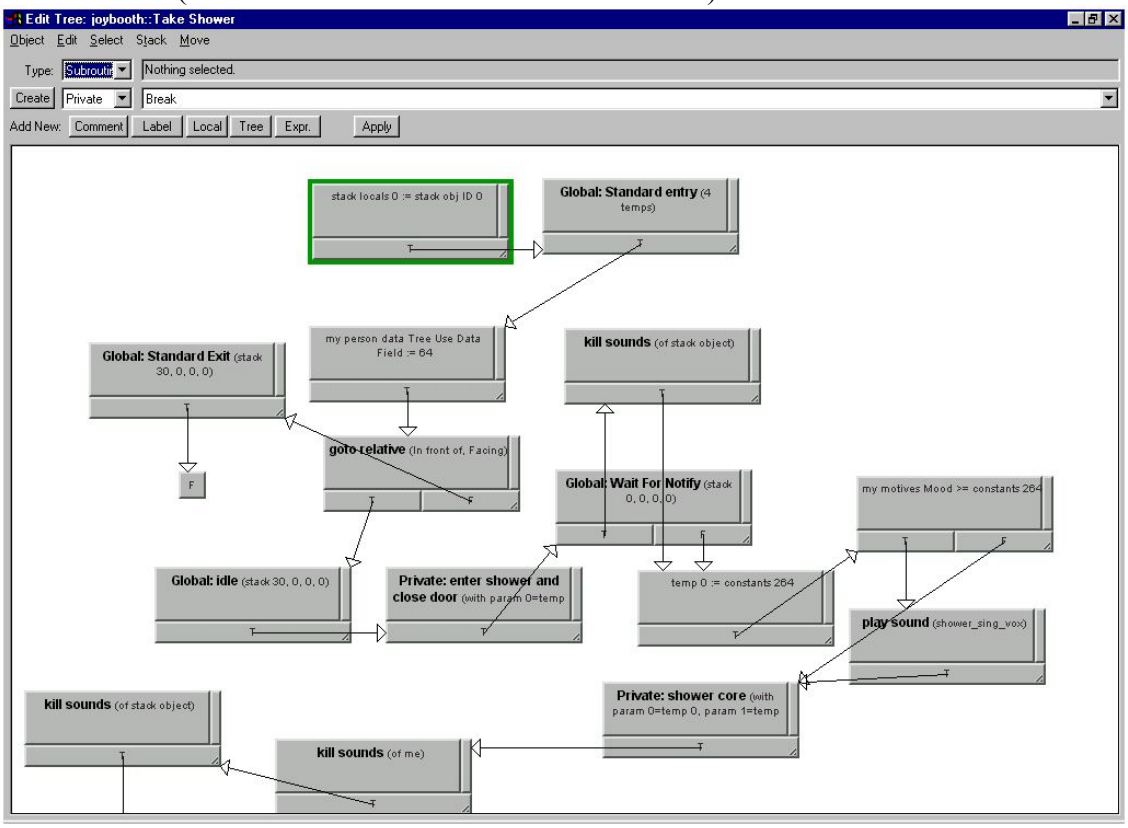

You'll see that we left the internal name of the routine (aka tree) to be Take Shower, but changed the way it presents to the player as "Take a Dose". We've made it available to everyone in the family and to visitors. It advertises strongly for Social, Fun, and Mood,

making it a good payoff for those circumstances where you might otherwise hang out with family, friends, and neighbors.

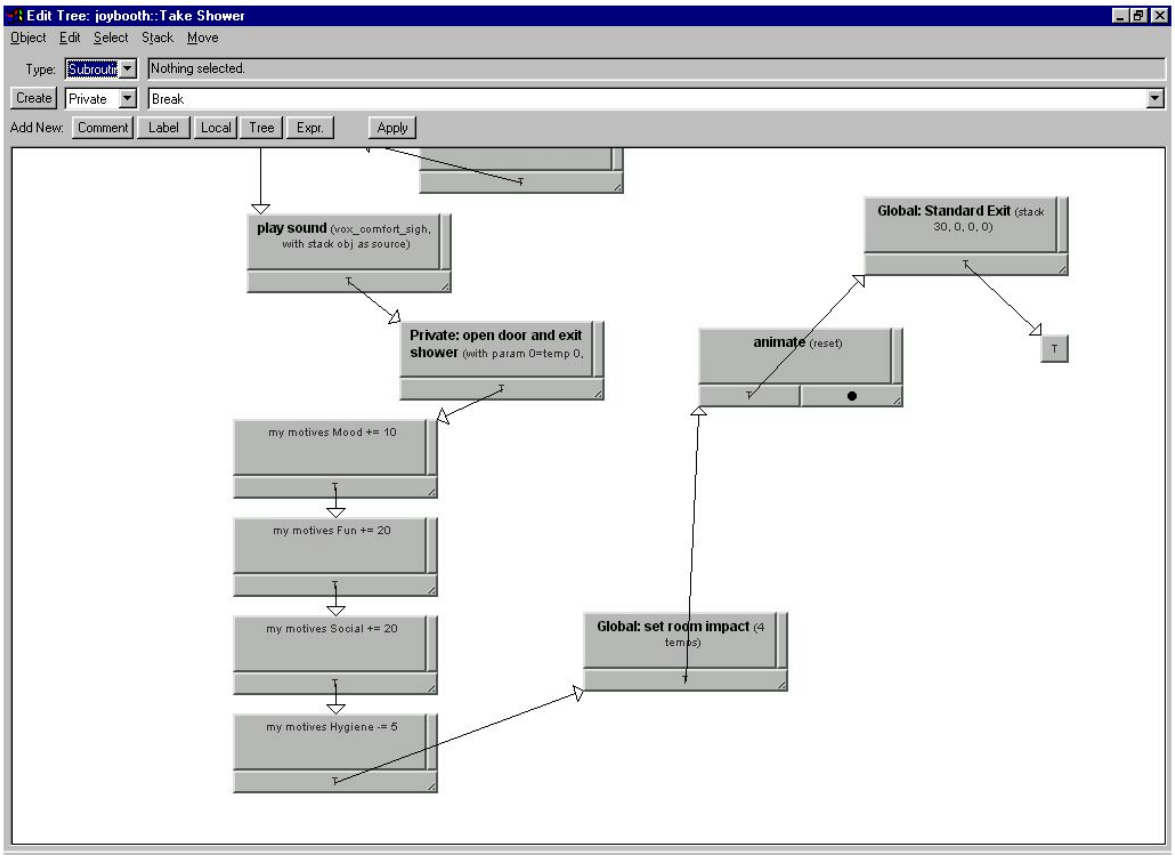
How should taking a dose work? It's useful to think of this in terms of differences from how the shower works:

- Showers can only be taken in private. We want the Joy Booth to be useable in public.
- Showers require taking off one's clothes. Since Joy Booths are installed in public places, we'll let Sims keep their clothes on when using a Joy Booth.
- A shower increases hygiene. We're going to increase Social, Fun, and Mood, and actually decrease hygiene a bit. (How many ways of having that much fun don't involve a little sweat?)

Fortunately, these changes for the most part can be done by stripping out parts of the Take Shower routine, specifically those parts which are concerned with disrobing and getting dressed again (see ) and worrying about privacy (see ). The simplified version of Take Shower (which to the user looks like Take a Dose) looks like this:




You'll notice that we first try to go to the front of the shower, facing it. If that fails, we exit. (It can fail because someone else could be in it, or blocking our route to it.) If it succeeds, we enter the shower and close the door (**Private: enter shower and close door**). Some magic with animation ensues, and if the mood is sufficiently high, singing in the shower is enabled. **Private:Shower core** runs the animations for actually taking a shower, and afterwards, some more cleanup occurs. What happens on exit looks like this:




You'll notice that once we exit the shower, we increment Fun, Social, and Mood, and decrement Hygiene slightly, before doing some cleanup operations on the way out (the Global set room impact, animation reset, and Standard Exit).

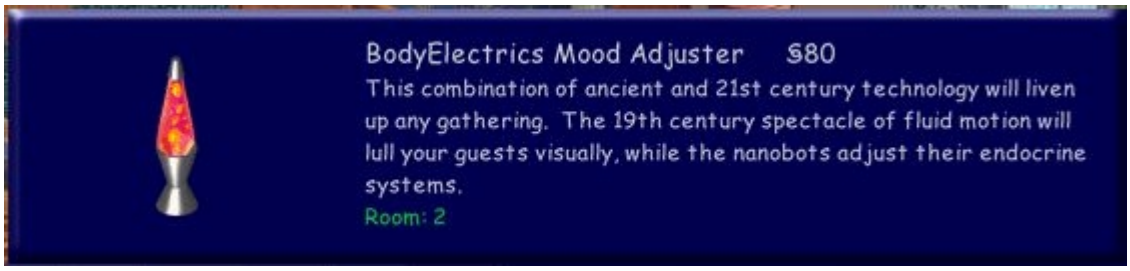
You'll notice that, viewed locally, the Joy Booth looks like a great object – if your neighborhood is sparsely populated or a Sim doesn't have very many friends yet, the Joy Booth will help them get through a day far more happily than they would be otherwise. Regrettably, as noted above, it is used at the expense of doing social actions that would build relationships. And thus in the end it probably isn't a terribly useful object to have around.

Creating addictive objects is relatively easy – give some object's behavior high advertisements along some dimension, and you'll see that behavior chosen over and over again to satisfy them[2]. What is more difficult is to create collections of objects that are balanced, where Sims interact with them in natural-looking patterns (or at least more amusingly varied than an obsessive-compulsive exhibits).

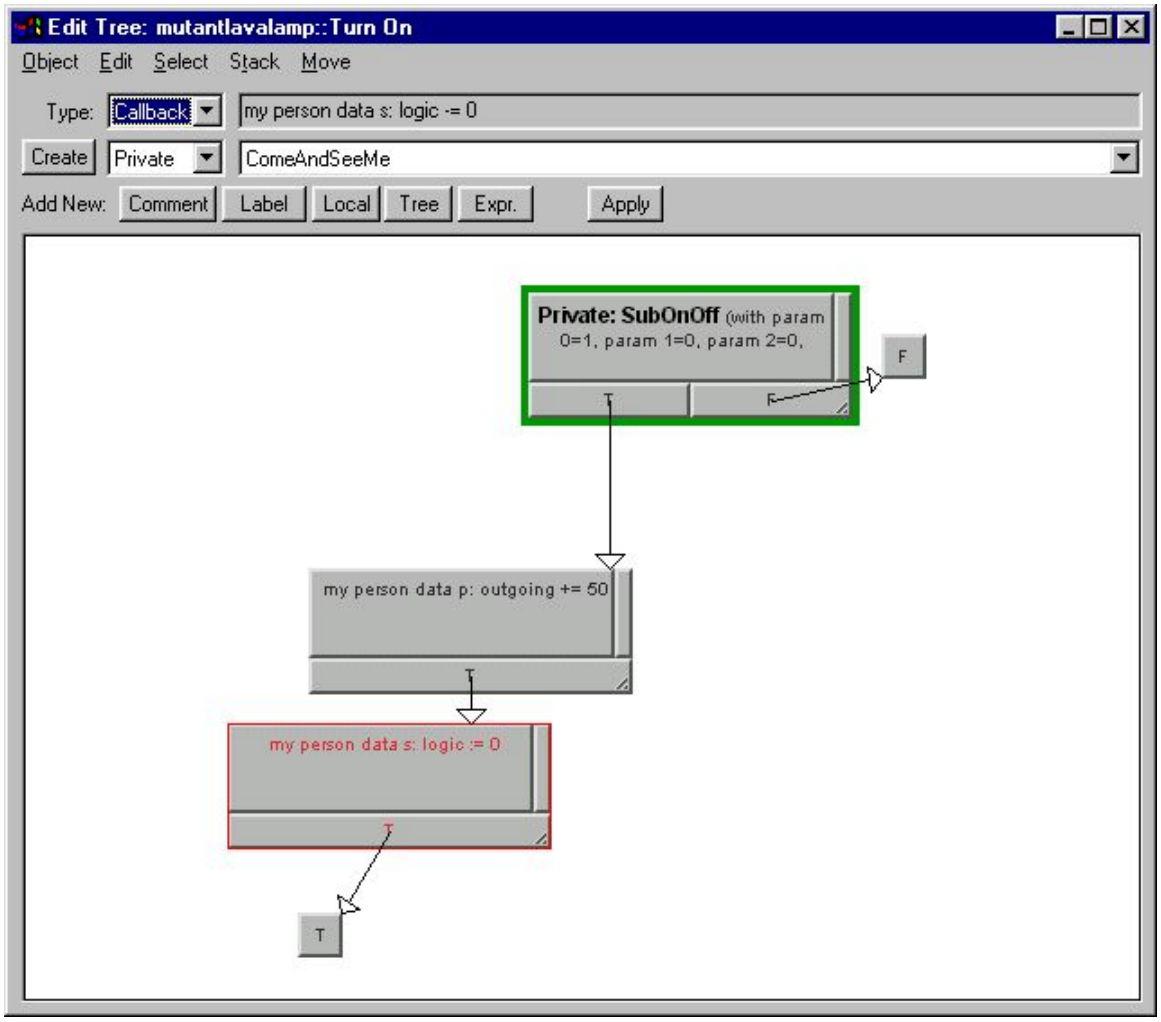
[2] Although even finding the right level of addition requires some tuning. Early versions of the Joy Booth were so addictive that Sims would continue using it until they collapsed. Not a very subtle nor very interesting, past the first time. However, it would be an interesting exercise to modify the Joy Booth so that (a) different personality types were more likely to find it very addicting and (b) they would build up a serious addition, by the use of a Relationship object, to it.

## 4.2 Exercise: The Mood Adjuster

The idea of a Mood Adjuster is to get everyone in a party mood. As the catalog entry states:




As you can see, the Mood Adjuster started out as a Transmogrified Lava Lamp. However, the Turn On behavior has been edited to represent the effects of the nanobots:




It first calls the **Private:SubOnOff** routine, which handles the Lava Lamp aspect of its behavior. But before exiting, it also makes two changes to the person doing the operation: It increments the Outgoing parameter of their personality by 50, and resets their Logic skills to 0.

As written, this routine has a couple of problems:

1. It seems a little extreme to make permanent changes to mood. To be sure, that might be the way nanobots adjusting one's endocrine system might work out in practice. But it seems doubtful that such a product would remain on the market very long. It would be better if the effects were temporary.
2. The effect is advertised in the catalog to be affecting everybody, not just the person who turned it on.

You might try as an exercise fixing the Mood Adjustor to overcome these problems. Here are some hints:

- The engine underlying the Sims doesn't have much in the way of compound data structures. However, there are a small number of unused parameters in the Person object that can be used as temporary caches. (This assumes that two object authors don't try the same trick at once, obviously.)
- In making the effects of the Mood Adjustor more global, there are two natural contexts: It affects everyone just in the same room it is in, or affects everyone in the house. Whichever one you pick, you'll need to iterate over all of the people in the room or house. If you look at the tree **Global::Privacy – test alone**, you'll see a method for iterating through people and seeing if they are in the same room.
- Visiting neighbors raise some interesting problems. They might come in after the Mood Adjustor is turned on, in which case they won't be affected, but might have very strange things happen if the Mood Adjustor is turned off while they are there and their state is "popped". They might be there when the Mood Adjustor is turned on, but leave before it is turned off, in which case the changes to them will persist. One solution is to live with it – nobody said nanotechnology was safe, did they? Another is to use a flag to indicate when someone has been Adjusted, and only pop values for those folks who have been Adjusted. (This doesn't help the neighbor-leaving problem, however.)

## *4.3 Some object suggestions*

In case you don't have some ideas you are itching to try, here are some suggestions to get you started:

- Nanobot Cleaners: Like the roaches, these critters swarm over your house. But they clean your house, including digesting trash on the floor. While expensive to purchase, they require no maintenance and even bring in a little income, because the trash that they digest is sold to the local recycling plant. (Finally, a reason to have an internet connection in your refrigerator ☺) However, at some infrequent interval, they will also swarm over and digest some object (or person?) in the household by mistake.
- Skeptical Sims: Objects advertise, to be sure, but currently Sims have no way of knowing whether or not the experience was worth it. Relationship objects are used sparingly in The Sims, but they enable recording information about a Sim's relationship to another object. Try storing all of the relevant variables in temporaries in a Relationship object before executing a behavior, and comparing them afterwards to see if

the object does indeed perform as advertised. Then use this information to scale the advertisements when evaluating future interactions appropriately.