SimAntics Data Holders – An Overview

An overview of Data Holders (variables) for SimAntics BHAV modding.

But first, some background information about objects and how they are identified in The Sims 2.

Object ID (or OID)

Most things in The Sims 2 are objects - Sims, pets, tables, beds, lights, cookers, food, plates, memories, disease tokens, noise emitters, controllers (eg, date, weather), jobs, schools, etc, etc, etc.

When an object is placed onto a lot - via Build/Buy Mode or by a Sim entering the lot - a copy of the object is created by the game and given a per-lot unique identifier - its Object ID (or OID)

No two objects on the lot will have the same OID. If there are two identical looking side tables either side of a bed, the bed will have its own OID and both side tables will have their own unique OID.

When a Sim enters a lot, a copy of the Sim is created from their character data file and placed on the lot, and they will have their own unique OID.

Note that OIDs can (and do) get reused - if one of our side table's OID is 123 and we delete that side table, the next object placed onto the lot may get an OID of 123. This is why it is a very bad idea to save OIDs as a reference to things across lot changes (going from home to community, or over Save / Load game sessions).

If you want to know what objects are on a lot, force an error on any object and look at the list of objects in the "ObjectError" log under the "Lot Object Dump" section. Anything on the lot not in that list is not an object (typically build items such as walls, fences, terrain paint, etc)

Neighbor ID (or NID)

Sims are different to other objects in that we need to be able to refer to them both across lot changes (home -> community -> home) and when they are not on the lot (picking a Sim to call on the phone)

Sims have a Neighbor ID (or NID) that is unique to the Sim within their master hood (and any attached sub-hoods - university, downtown, vacation destinations, secret hobby places, etc). If you need to refer to a Sim across lot changes, use their NID.

Objects Perform Actions

The Sims 2 is an action-based game - objects don't just sit statically on the lot, they do things (even if that’s metaphorically twiddling their thumbs). Some actions are user directed, for example, selecting an item from a menu for a Sim to perform, but most are game directed, for example, a Sim "autonomously" picking a menu item to perform or the game depreciating an object's value or making it snow or sending the job pool car.

An object performing an action is executing code, usually within a BHAV but occasionally via a Lua script.

Object Types and Object Instances

What we as modders refer to as an object, in strict programming terms is actually an object type, that is, the object's definition - name, category, price, appearance (mesh and textures), configuration (BCONs), code (BHAVs), etc. For a custom object, this is basically everything in the .package file.

An object instance is a single, on-lot manifestation of an object type. So, for our two side tables there is one object type (the definition of the side table) and two object instances (each placed side table). Each object instance maintains its own state, for example, its current value (depreciated price), and where in the code it is (one fireplace may be cold, while another is setting light to the curtains!)

Object Lifecycle

When an object is placed onto a lot, the game creates a new Object Instance from the Object Type, assigns it a unique Object ID and then executes its Init Function (as determined from the object's OBJf init definition). Assuming the Init Function BHAV returns true, the game then executes the object's Main Function (as determined from the object's OBJf main definition). And that's it, the Main Function should "idle-loop" until the object instance determines that it should delete itself (if ever). If the player deletes the object from Build/Buy mode, the game will abort the Main Function.

Why "idle-loop" is important

In a modern game with multiple objects, there is an over-arching manager that gives each object a small slice of time to execute its code before it is stopped, and the next object given some time. This is known as "pre-emptive multi-tasking", multiple objects doing tasks with a manager that hands out small slices of time to the objects in order and pre-empts them when their time is up.

The Sims 2 does NOT work this way. It uses a scheme known as "co-operative multi-tasking", where each object is responsible for pausing its own execution on a regular basis and co-operating with other objects by letting them execute for a bit. In The Sims 2, the "pause and let some other object run" is handled by the sleep primitive, which is wrapped by the various Idle globals. (You should never call the sleep primitive directly, but always use one or other of the global Idles.) If an object is coded in such a way as to infinite loop without calling Idle, the game would freeze - known as "dead-locked". To stop this, the SimAntics interpreter counts the number of primitives executed between calls to sleep and if the count exceeds 10,000 assumes the object is dead-locked and causes the object to generate an error. Note, some long running processes, such as building the list of all Sims in a hood for the phone book, could validly exceed 10,000 primitives, which is why such code is written to process the Sims in blocks (for example, 50 at a time) and idle between each block.

In a BHAV, what is "My"?

The "My" data holder (variable) is the currently executing object instance, if you know any "Object Orientated" (OO) programming language it is the equivalent of "this" (or possibly "me"). Typically, it is the OID of the Sim performing an action. But it can also be the OID of an object instance (as all objects perform actions), for example, in the Init and Main Functions "My" will refer to the object. Importantly, you CANNOT set the value of "My" to refer to another Sim or object, it is ALWAYS the Sim or object that is executing the code.

What is "Stack Object ID"?

The "Stack Object ID" data holder (variable) is whatever we want it to be - usually, but not necessarily, an Object Instance's OID. As such, it typically indicates the object instance a Sim is interacting with.

Other Data Holders - Assuming "Stack Object ID" contains a valid OID - object or Sim

My / Stack Object's - information about the object instance, useful ones are "containerid" (am I within another object), "slot number" (if within another object, which slot of the containing object), "room" (which room I'm in - 0 is outside, -1 is off-world), "graphic" (what an object looks like) and "object id" (my OID)

My Attribute / Stack Object's Attribute - values specific to the object instance, for example, who a driving licence is owned by and the number of points on their licence.

My [semi] Attribute / Stack Object's [semi] Attribute - values specific to the object instance from their associated semi-globals, for instance, how dirty a shower is, or if a light is on/off.

Stack Object's definition - values from the Object Type this object instance was created from, this permits access to the data in the OBJD for an object.

Other Data Holders - Assuming "My" / "Stack Object ID" contains a Sim's OID

My person data / Stack Object's person data - general personal data for the Sim, for example, skills, interests, job, age, zodiac, etc, etc

My motive / Stack Object's motive - motive data for the Sim

Other Data Holders - Assuming "Stack Object ID" contains a Sim's NID

Neighbor in stack object - some info about the Sim and their home.

Neighbor's person data - general personal data for the Sim.

NIDs, OIDs and Log Files

There is NO relation between NIDs and OIDs, specifically a Sim does NOT get their NID as their OID, a flowerpot could have the same OID as a Sim's NID.

Now this can cause confusion when reading error logs as the log file assumes that the contents of the Stack Object ID is always an object, even when it's a Sim's NID. Which leads to the interesting log files where a Sim appears to be making a phone call to a flowerpot!

=== END ===