Bethesda Tutorial Quest Aliases

creationkit.com/index.php

Contents

Overview

This chapter will teach you about more advanced uses of aliases, showing the full use and power of these crucially important data structures.

The reader will learn:

- How to use conditional aliases
- How to properly organize quest logic with aliases

Doing It Wrong

Up until now, we've been doing it wrong.

Well, not completely wrong, just in a way that scatters our logic all around the master file -- this makes it difficult to track down problems and make changes. (Software engineers would call this bad encapsulation.)

Consider: Bendu presently has a script on him that sets stage 200 on GSQ01 in the event of his death. This works fine for our isolated circumstance, but imagine if Bendu was not someone we had created just for our quest. If another quest wanted to use him, the designer would potentially have to modify our script, or add another script. If he was a normal character in the world, he also wouldn't have "GSQ" in his name, which means it might be difficult to track down "where is the script that's failing my GSQ quest?"

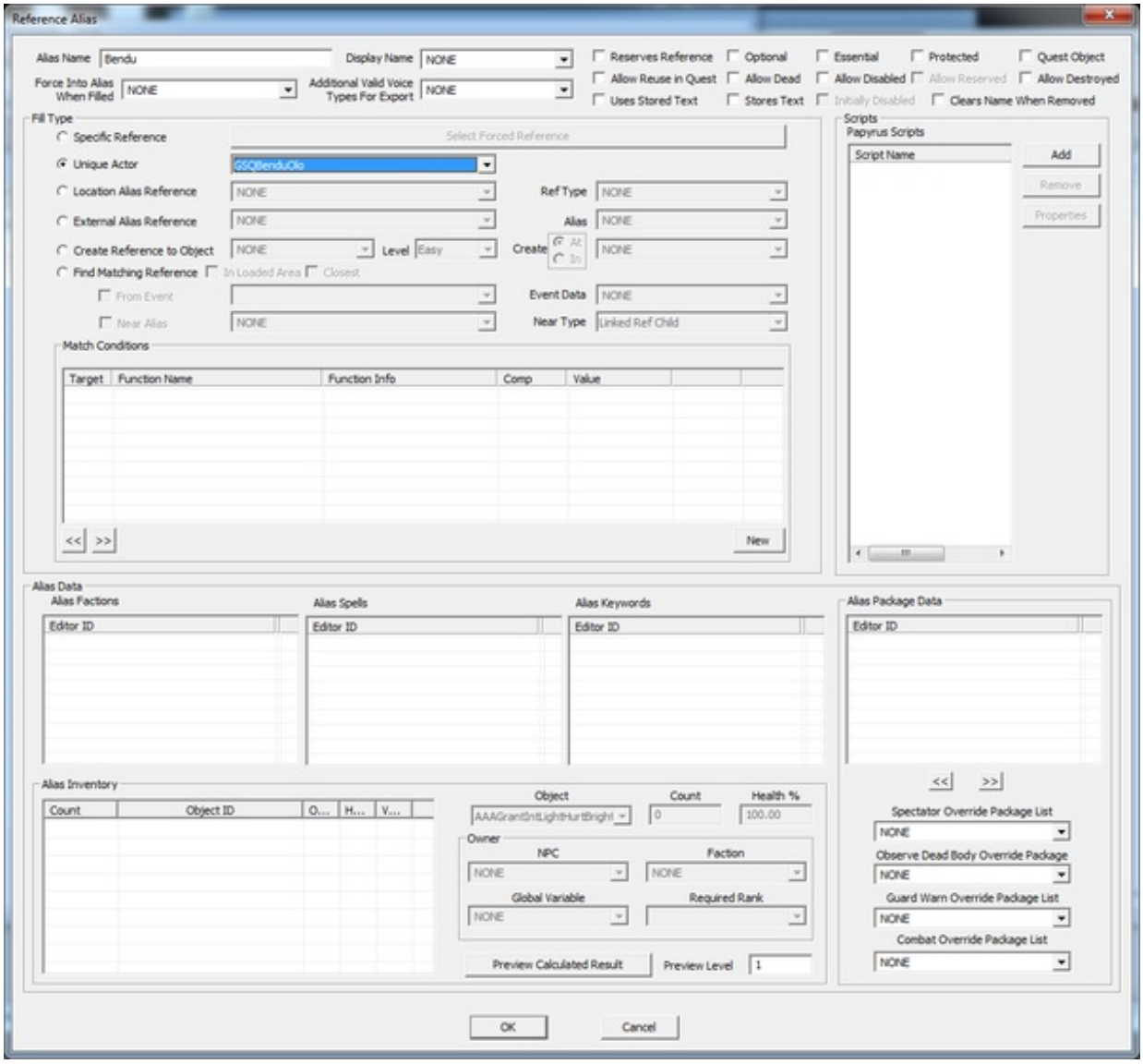
The same problem applies to the amulet, and the thief. Obviously our current setup works well, but wouldn't it be nicer to have access to all the things our quest cares about in one place? Well, with aliases, we can have just that!

Aliases as Overlays

The best way to think of aliases is as roles, defined for the specific purposes of their given quest. Characters and objects take on these roles for the duration of the quest, and then can shed them when the quest is done. This leads to much cleaner design implementation as well as memory savings.

Open up the GSQBenduOlo actor that we made towards the beginning of the quest design tutorial series. After we tidied up the loose ends, he either has a script on him to handle his death, or has been flagged as essential. Both of these are very quest-specific bits of information though -- ask yourself, if the GSQ01 quest did not exist, is there any reason for them to be there? Since the answer is "no," clear them off of the base object: uncheck the "Essential" flag, and use the "Remove" button in the scripts area to take the script off of the base actor.

Now open the "Bendu" alias we made during the Quest Design Fundamentals chapter on objectives. Then, we were studiously ignoring most of this rather large window and all its various doodads, but now we'll actually look around a bit. Don't be scared.



So looking around here, we see a lot of familiar elements. There's a script panel. A package stack. An inventory. _Et cetera_. We can treat these as we would the same areas on a base object, and when an actor is put into an alias, he will take on all of the data for that alias as if they were his own. More importantly, when a quest stops running, all its aliases are removed from their targets.

This is an important concept, so worth clarifying and mentioning again: **An actor only takes on the data from an alias while its quest is running and he is in that alias.** Once the quest stops, the actor will shed the alias like taking off a coat. (It's also possible to clear an alias, moving the actor out of it, while the quest is still running.)

This has a number of benefits:

- Our scripts can be specific to this individual quest, which makes them simpler and thus easier to debug.
- We can make quest-specific packages that will only govern the actor's behavior during the quest. (Note: the alias package stack sits *on top* of the actor's normal package stack, so alias packages will have priority.)
- We can apply spells (including passive abilities) and factions for the duration of the quest, which can have dramatic impact on how an actor behaves, especially in combat.
- If the quest changes and we decided to use a different character in the world, we only need to change the target of this alias and **everything just works.** It's kind of delightful.

To make it easier to swap alias targets after the fact, it's good practice to make dialogue conditions use GetIsAliasRef instead of GetIsID. In this case, we'd want to go back through all our dialogue and replace our "GetIsID GSQBenduOlo == 1" conditions with "GetIsAliasRef Bendu == 1".

One "gotcha" with the overlay concept is the inventory: any items you put into an actor's inventory will stay there when they drop out of the alias. (You could, of course, remove things with a script if you need to, but consider the ramifications carefully.)

On most computers, the "Reference Alias" window is so tall that you won't be able to reach the bottom of it to click the "OK" button. To get around this, you can click back in the text box for "Alias Name" when you are all done editing and press the ENTER key on your keyboard. This will save the work done in the window.

Fixing Bendu

But enough talk. Let's fix Bendu's alias so that his quest functionality lives here. As before, we have two approaches:

Immortal Bendu

You can simply click the Essential checkbox at the top of the Alias window, and Bendu will now be unkillable while he is in this alias. When the quest stops, he'll drop out of the alias and again be mortal.

Handling the Death

Or we can put a script on this alias to handle Bendu's death, just like we had previously scripted the base object. Note that we have to change the first line of the script so that it will attach to a ReferenceAlias instead of an actor. The script we would make this time would be called "GSQQuestgiver" and look like this:

```
Scriptname GSQQuestgiver extends ReferenceAlias

Event OnDeath(Actor akKiller)
 if (!GetOwningQuest().IsCompleted())
  GetOwningQuest().SetStage(200)
 endif
EndEvent
```

Note that we don't have to set the quest as a property this time around; every ReferenceAlias is aware of the quest that owns it, so we can access it with `GetOwningQuest()`.

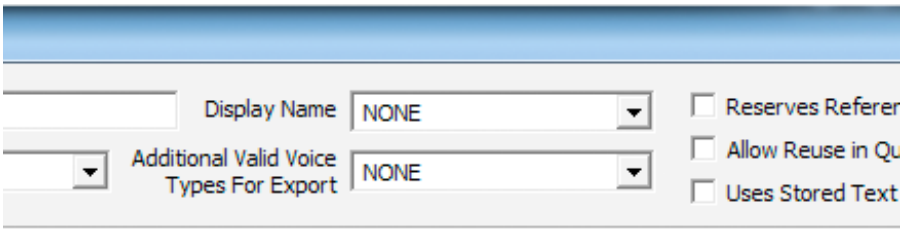
Now, setting a stage when an Actor dies is an exceedingly common task, so there's no need to make a specialized script for it. When you click the Add button to put a script on this alias, look for scripts that begin with "default" -- these are provided scripts that perform common tasks. They are generally named pretty descriptively, so you can guess what they do from what they're called. In this case, we want "defaultSetStageOnDeathRefAlias". In this case, we'll need to set properties:

- myQST: GSQ01
- preReqStage:
- StageToSet: 200

(By using default scripts, we can drastically decrease the total number of scripts in the game, which has benefits for memory usage. Since you're modding for a PC that likely has copious memory, it may be less of a concern.)

Changing the Name

One final side note: you can even change the name of a reference when it's in an alias, so that players will see different text when they roll over it. You can define a Message with whatever text you want, and select it from the Display Name pulldown at the top of the alias window.



By default, once you place a new display name on a reference, it will retain it *even after your quest is over*. This is so the player doesn't come back to a cleared-out dungeon to find that all the enemies are now called different things than they were when they were killed. You can override this behavior with the "Clears Name When Removed" checkbox in the upper right.

Packages

The package stack for the alias will be placed **above** an actor's normal package stack, effectively overriding it.