Bethesda Tutorial Quest Loose Ends

Contents

Overview

This tutorial will wrap up the loose ends we've left dangling as we've constructed the epic tale of Bendu Olo and his lost amulet.

This final tutorial is much less hand-holdy, and assumes that you've absorbed the material from previous tutorials in the Quest Design Fundamentals series. If you need a refresher on anything here, you should be able to find what you need in the previous chapters.

The reader will learn:

- What we've done wrong.
- How to fix it.

Problems

There are a number of issues with the quest as we've implemented it.

1. Bendu can be killed after we accept the quest, which would render it impossible to complete if we've already started it.
2. We don't add the map marker to the player's map when Bendu tells where to go.
3. It's possible for the player to get the amulet via pickpocketing, which would advance the quest properly, but leave extra objectives hanging around uncompleted.
4. The player could find and kill the thief before meeting Bendu, which would cause the quest to advance, and be confusing.
5. Bendu doesn't actually take the amulet from the player at the end of the quest.
6. The player can drop the amulet and then lose it, which would be her own fault, but still something we should think about.
7. If the player chooses not to help Bendu, he won't acknowledge having met the player before, which can be a little jarring.
8. Bendu just has generic hellos and goodbyes, rather than acknowledging his specific circumstance.

We'll go through these one at a time -- some of them have multiple solutions, with pros and cons.

Killing Bendu

We have two options for handling Bendu dying: we either shut down the quest so it doesn't hang around annoying the player, or we make it so Bendu can never die. Obviously both sides have their pros and cons -- if Bendu is a character we need in the future, we may want him to be unkillable anyway, for example.

Handling the Death

We've already shown how to complete a quest at a certain stage (with the Complete Quest) box. When that stage is set, the quest gets moved to the inactive part of the player's journal and the "COMPLETED: <Quest Name>" banner will pop. Failing a quest works similarly, but the player will see "FAILED" instead.

Make a new quest stage numbered 200. Create a log entry for it with some downer conclusion text:

> *I failed to return Bendu Olo's amulet to him.*

Check the "Fail Quest" box below it.

Secondly, we need to set up Bendu to set this stage when he dies. Put a script on him, (like we did with the thief) called GSQBenduOloScript. Here's the text of the script, after the initial Scriptname line:

```
Quest Property GSQ01 auto

Event OnDeath(Actor akKiller)
 if (!GSQ01.IsCompleted())
  GSQ01.SetStage(200)
 endif
EndEvent
```

Note: we check to make sure the quest is not already completed, because we wouldn't want the player to fail after finishing it.

You can auto-fill the GSQ01 property,

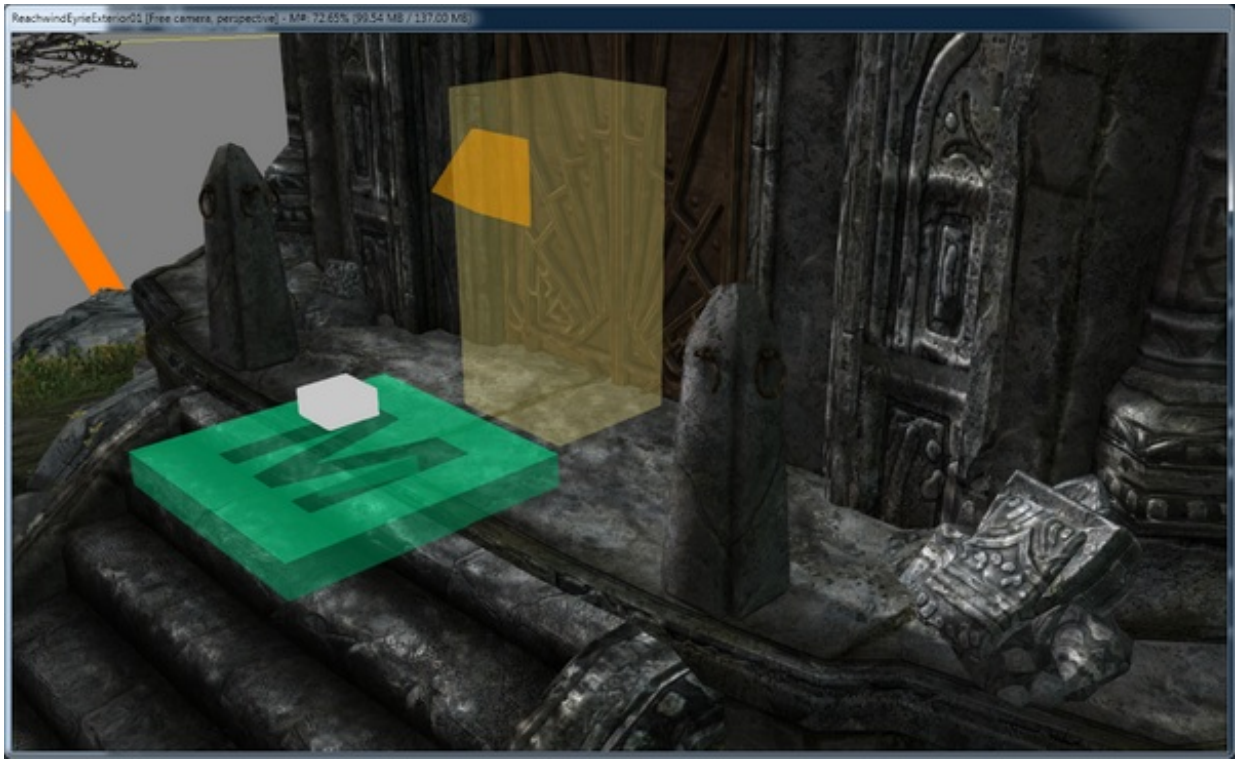
Immortal Bendu

The other option is to make Bendu unkillable -- if the player attacks him, he will go into bleedout (writhing on the floor), but stand up again when combat is finished.

This is remarkably easy to accomplish. Open up the actor window for GSQBenduOlo, and click the "Essential" checkbox under the name fields. ("Protected" is a similar state, but it allows the player and only the player to kill the actor.)

Map Marker

Typically, when we tell the player to go to a place in the game, we add the map marker so they can have an easy target to head towards. Navigate to the cell called ReachwindEyrieExterior01 in the Tamriel worldspace. (You can get there easily by double-clicking the yellow door marker inside ReachwindEyrie01.) You'll see a green-blue rectangle with a large M carved out of it right next to the exterior door marker.



Open the GSQ01 Quest and click on the Scripts tab, add an ObjectReference property to the script, and call it "DungeonMarker". Select the map marker in the render window as the target of the property.

Now, edit the stage 10 script so it reads:

```
SetObjectiveDisplayed(10)

DungeonMarker.AddToMap()
```

Pickpocketing

If you think about the script we put on the amulet to advance the questline when the player takes it:

```
Quest Property GSQ01  Auto

Event OnContainerChanged(ObjectReference newContainer, ObjectReference oldContainer)
 if (newContainer == Game.GetPlayer())
  GSQ01.SetStage(30)
 endif
EndEvent
```

This means that when the player picks up the amulet, it will always set stage 30 of the quest. If this happens without killing the thief, stage 20 will never have been set, and so we will display the next objective (to return the amulet to Bendu) without completing objective 20.

The best way to handle this is to have stage 30's script check to see if stage 20 has been run, and handle the objectives properly. So the old script simply looks like this:

```
SetObjectiveCompleted(20)
SetObjectiveDisplayed(30)
```

And we'll change it to look like this:

```
if (!GetStageDone(20))
 SetObjectiveDisplayed(10, False)
endif

SetObjectiveCompleted(20)
SetObjectiveDisplayed(30)
```

That second line of code is setting the objective to **not** be displayed, so it will just quietly disappear from the player's journal.

(Of course, we could also simply never have the "Kill the thief" objective, and just tell the player to retrieve the amulet. This is probably better from a design perspective, as it doesn't encourage any particular playstyle over another.)

If the player kills the thief after pickpocketing the amulet, the player will receive an additional objective; to retrieve the amulet we already have, leaving two active objectives. We can easy solve this by editing the TutorialQuest script we gave the thief.

```
Event OnDeath(Actor killer)
 if (TutorialQuest.GetStage() < 30)
        TutorialQuest.SetObjectiveDisplayed(20)
        TutorialQuest.SetStage(20)
 endif
EndEvent
```

This will check if the quest stage is below 30 before it changes the stage for 20. So if player already pickpocket the amulet changing quest stage to 30, then killing the thief will never trigger stage 20.

Out of Order

Of course, an even bigger problem is that the player could just stumble into Reachwind Eyrie without ever having met our friend Bendu. This would complete objective 20 (which is undisplayed, so the player won't see anything) and display objective 30 to return the amulet to Bendu. At which point the player might say "Wait, what? Who? I've never met this person!" There will also be no journal update until the quest is completed, and it will just be an all-around confusing experience.

Again, we have a few options here:

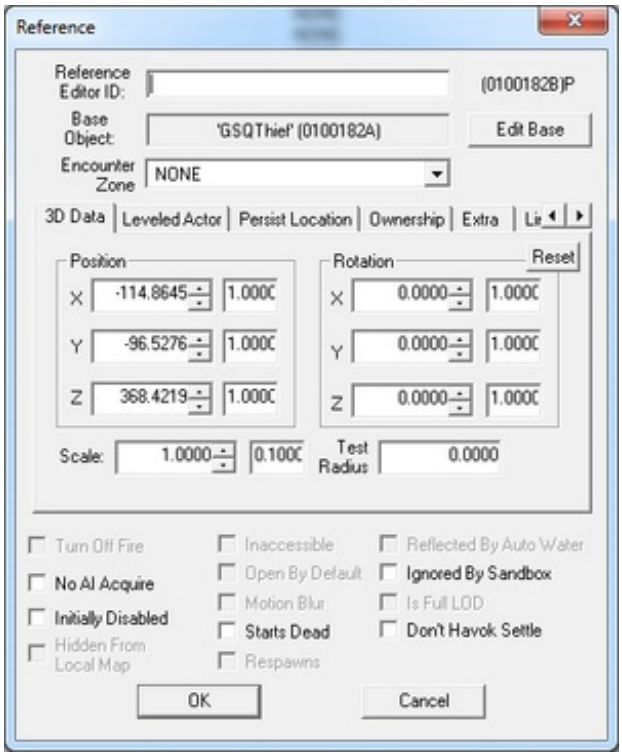
1. Don't create the amulet in the thief's inventory, but put it elsewhere in the world (like an inaccessible chest). Then, when the player accepts the quest, resurrect the thief if necessary and move the amulet to the thief.
2. Set the amulet to be the thief's death item (an object that is only placed into the inventory upon the actor's death so that it can't be pickpocketed). Then resurrect the thief (if necessary) when the player accepts the quest.
    - (The amulet would have to be set as a conditional death item, so it would only get placed there if we were at the appropriate quest stage.)
3. Allow the player to kill the thief early, but replace the option to help Bendu with a line saying "You mean... THIS amulet?!" and completing the quest right there.

4. Leave everything as it is, but disable the thief until the appropriate stage, so the player will never have the chance to kill and loot before the appropriate time.

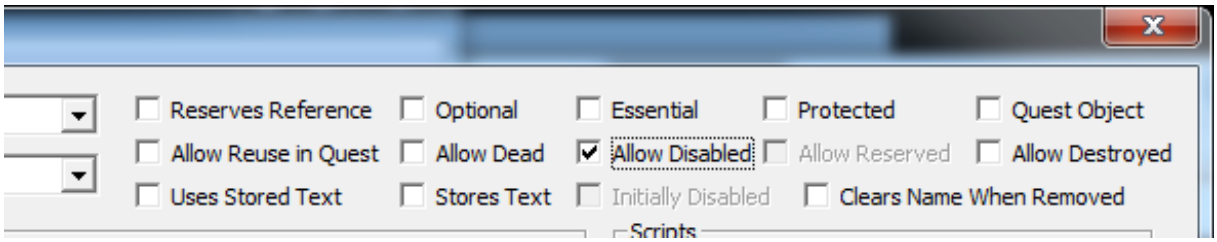
In our scenario, the final option is by far the simplest and most reasonable one, but the other two are given to illustrate the kinds of logic which sometimes need to be used to accommodate a particular bit of story happenings.

To disable the thief, simply double-click on the reference we placed in ReachwindEyrie01. This will bring up the Reference Window.

Simply click the checkbox for "Initially Disabled," click OK, and the thief will not be at Reachwind Eyrie when you start the game. Try it!



Before we go any further, open up the Thief alias that we made (in the Quest Alias Tab of GSQ01). Click the "Allow Disabled" checkbox that's nestled in the large field of checkboxes in the top right. This will allow the alias to point to a disabled reference.



Technically for a "Specific Reference" fill type alias, you can get away without checking this box, but it's good practice and will help establish strong alias habits for when we do the more advanced material later. :-)

Now we just need to set the thief to turn on at the appropriate time. We'll do this in the script for quest stage 10. Alter it so it now reads:

```
SetObjectiveDisplayed(10)

Alias_Thief.GetReference().Enable()
```

The first line was already there, but the second line takes some explaining.

- `Alias_Thief` -- This is a property that the editor created and filled for us. Every time you make an alias, the game will make a property on the quest script called Alias_. This is a great convenience.
- GetReference() -- This is where things can get a little confusing, but the crux of it is that an alias is **not** the thing it points to. The alias itself is just a role that the quest knows about, but it has no idea whether that points to an actor in the world, a piece of wall geometry, a chest, etc. To get access to the thing the alias points to, we call this function.
    - (If this is confusing to you, don't fret. We'll explore aliases in more depth later on. You can just copy+paste the code here and move on happily.)
- Enable() -- Actually enables the reference.

Now when we play the game, the thief will be missing until the player accepts the quest, at which time he will begin his thieving existence.

Giving the Amulet to Bendu

Right now the player neither gives the amulet to Bendu nor receives any reward when completing the quest. Both of these are easily fixed.

Open up the stages tab again, and go to stage 40. Right now its script only contains `SetObjectiveCompleted(30)`. We'll add additional logic below this.

The first line we'll put in is:

```
Alias_Bendu.GetReference().AddItem(Alias_Amulet.GetReference())
```

Note that with `AddItem`, you can give it either a base object (in which case it will create a new instance of that object out of thin air in someone's inventory), or a reference (in which case it will move that reference from wherever it is). So this line will grab the amulet out of the player's inventory and put it into Bendu's. Neat!

Now there's the matter of the gold. We add gold the same way we add any item. But that means we have to make a property on the quest script that points to the gold. Click on the "Properties" button, and you'll see the automatically-created alias properties already in the list. Add a new property of type "MiscObject" and call it "Gold001" -- it will auto-fill, so you're good to go.

Back in the day, we used to have different base objects for different denominations of gold, hence the "001" appended to the name of the single gold piece.

Now that we have that property, we just have to add this line:

```
Game.GetPlayer().AddItem(Gold001, 500)
```

(Note that if you don't give `AddItem` a number, it assumes you're only adding one.)

Now, Bendu promised the player twice what the amulet was worth, and we're delivering that with the 500 gold. However, say we have to change the value of the amulet for lore, balance, or whimsical reasons. Then we would also have to remember to come back here and change the reward value. Or we could be slightly more clever about it.

```
Game.GetPlayer().AddItem(Gold001, Alias_Amulet.GetReference().GetGoldValue() * 2)
```

Now it will automatically multiply the worth of amulet by two, and give that much gold to the player.

> Note that we're making this gold out of thin air, rather than having it in Bendu's inventory and moving it to the player. This is a minor break in "realism" that we typically use for quest rewards, to avoid players getting them too easily via pickpocketing/killing/etc.

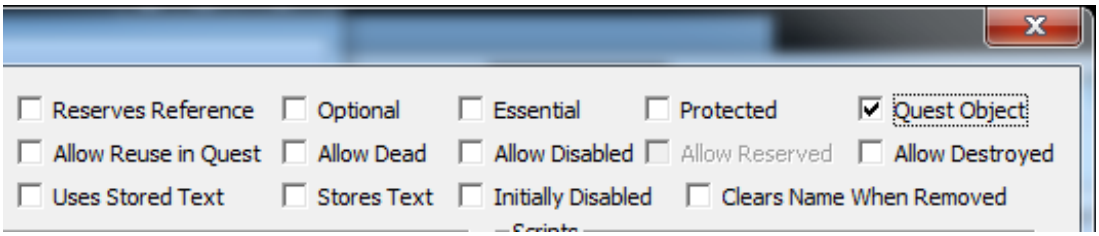
Dropping the Amulet

Players in Elder Scrolls games tend to be packrats. They pick up everything they can until they're overburdened, then they drop everything that doesn't look important. Or maybe they want to use the amulet to decorate their house. Or they throw it behind them hoping it will distract a giant as they run away. (Spoiler: it won't.) All sorts of reasons players drop items, but it can lead to problems when they go to finish this quest and don't know where the item is!

We could do **all** sorts of things to handle this (turning the objective to retrieve the amulet back on if they lose it, inserting a radiant reaction to have someone bring it back to you, etc.). For extra credit, you try to set up that first one; you have all the tools to do so now!

But the simplest and by far most foolproof solution to this problem is to simply make the amulet into a quest object. This means the player won't be able to drop it or sell it, and it will never leave her inventory until we take it out with the script to remove it.

We set this up with a flag on the alias, so open up the Amulet alias that we made long ago, and click the "Quest Object" checkbox in it.



Now the player will be stuck with this item until they finish the quest. (Luckily, quest objects don't count towards encumbrance; so it will never be a true burden.)

> In Fallout 3 and earlier Elder Scrolls games, the "Quest Object" flag lived on the base object itself.

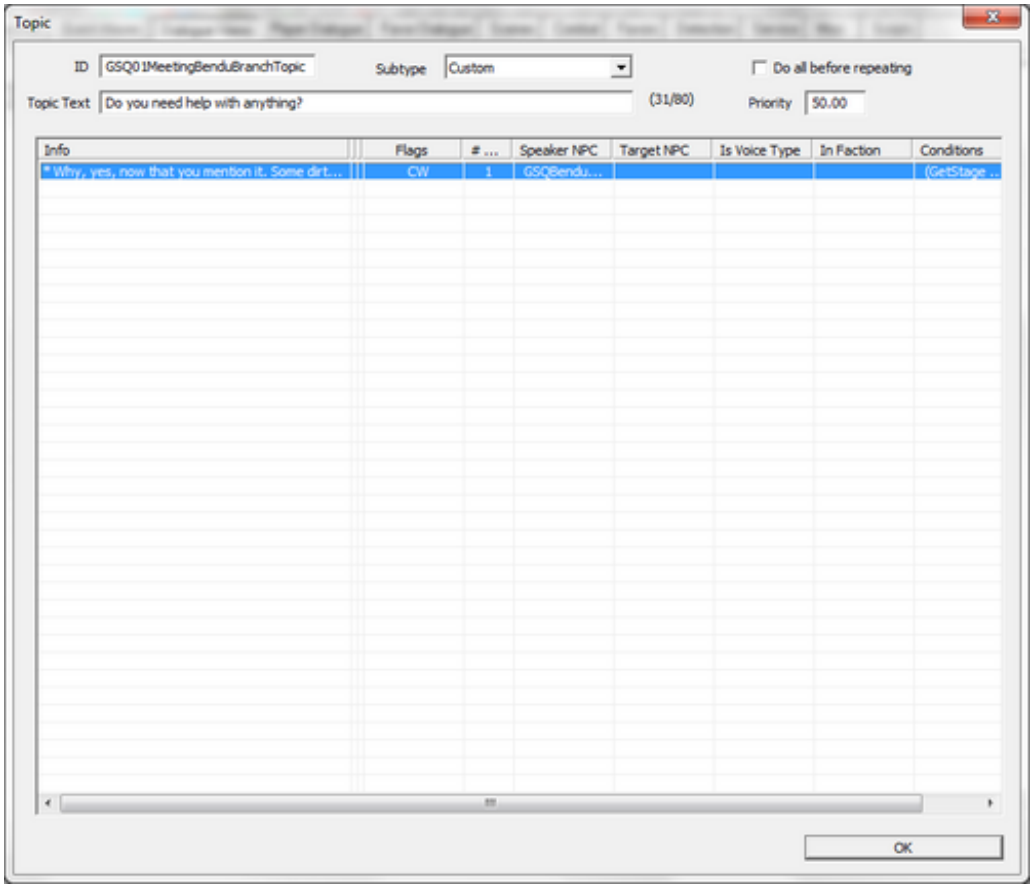
Coming Back Later

Then we have the issue of the player coming back to Bendu and him having no recollection of meeting the player previously. He'll give the whole sob story again. In this example, it's just a single line, so it isn't too bad, but if there was a whole line of questioning that led up to this, the player wouldn't want to traverse it again. So we'll add some handling for the player who said "no" and decided to come back.

The first thing we'll do is add another stage to the quest, stage 5. We won't have to add any journal entries or quest logic here -- this is purely for our own internal usage.

Then, if the player chooses the "No" option in dialogue, add a line of script to that info that sets the quest stage to 5. (Look at the "Yes" branch if you need to cheat and see how we set it to 10.)

Now comes the fun part. Open the initial topic (not the info) GSQ01MeetingBenduBranchTopic. You should be looking at this window:



Right-click on the highlighted info in the table and select "Copy". This makes a full duplicate of that info -- if it had a script, the new info has its own script containing the code that the first one did (so you can change one without affecting the other); if the first one had connections, the new one will have the same connections; if the first one was flagged as Goodbye, this one will be too. And so on.

When the dialogue system comes to a topic (i.e., when the player chooses the "Do you need help with anything?" text from a topic list), it starts at the top of its list of infos and works its way down, looking for one that has valid conditions. This is how you could have multiple characters responding to the same topic differently (by using different GetIsID conditions), or, in our case, having a character say something different at various stages of the quest.

Open the top info, and change its GetStage condition from "< 10" to "== 5". Now the first time the player chooses that topic, the game will check the first info, see that it's invalid (because the quest stage is 0), and then move onto the next one, which will be valid.

Change the response text to be something suitably snarky about the player taking his sweet time in helping. Example: "I already told you about my amulet. Are you going to help me or not?"

We can also fill in the "Prompt" field at the top of the window. If this is filled in, the prompt will override the topic text. By using prompt overrides, you can get a more specific back-and-forth feeling to dialogue.

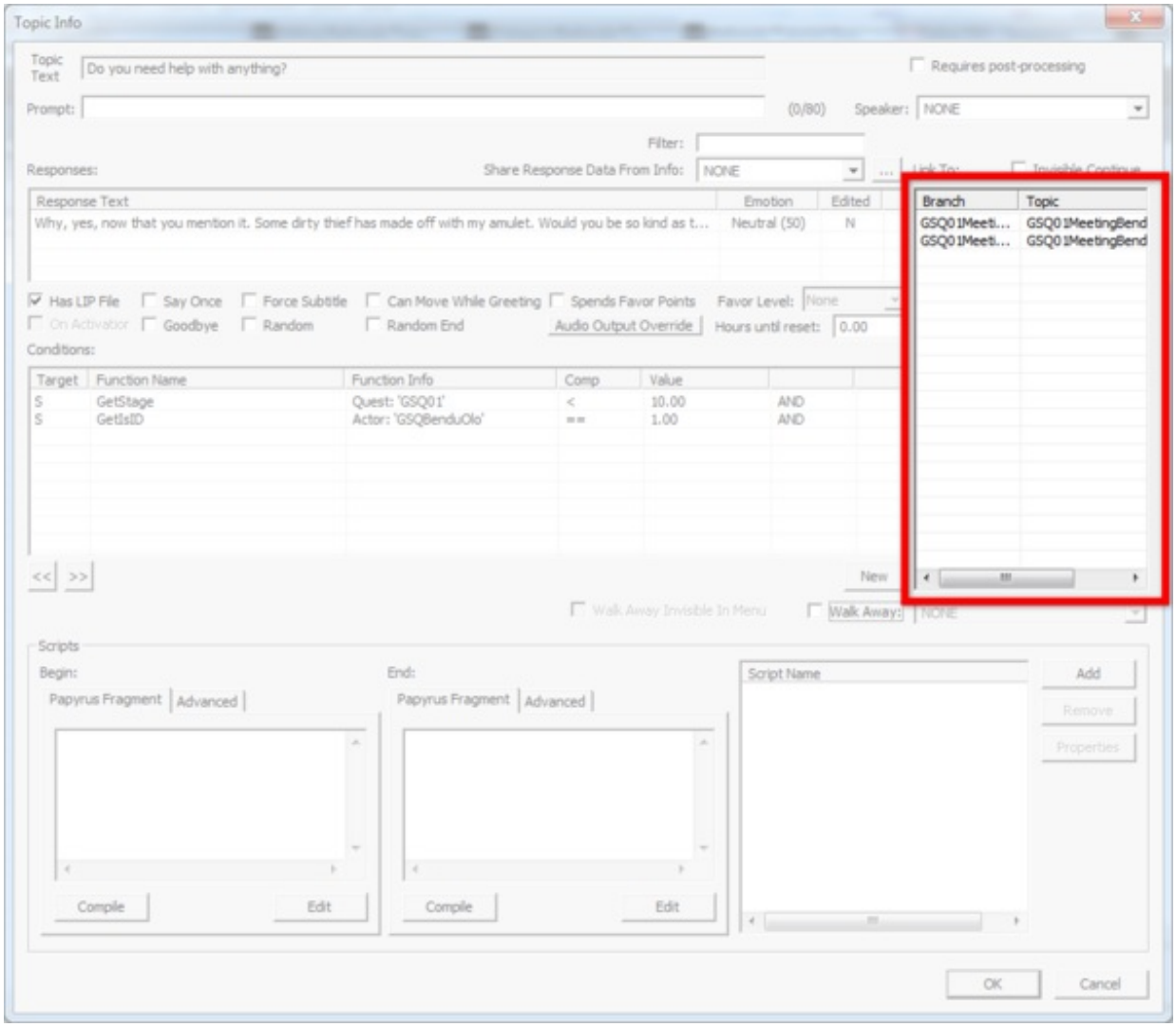
So fill in the prompt with "Didn't you need something?"

One final adjustment we'll make here -- in Skyrim, we introduce the ability for a player to dismiss themselves from a conversation at any time, without necessarily choosing an option in dialogue. For certain conversations, though,

walking away **is** expressing a choice, and the NPC should respond to that. (It's very important when you're talking to a guard trying to arrest you, for instance.)

In our case, we'd want Bendu to treat the player walking away the same as he would if the player actively said "No."

So open up the **other** info in this topic, the one where we didn't make a new prompt. Over at the right, you see a list of other topics that this one links to.



When you link something in a dialogue view, it shows up in here. (If you're old school, you can actually make the links from here, too!) Underneath, click the checkbox that says "Walk Away:" and select the GSQ01MeetingBenduNo link.

Now if you leave the dialogue while Bendu is talking, he'll react just as if you had chosen "Sorry, not right now."

We could also have handled this by playing with the reset timers on the initial line, or using a quest variable instead of a stage.

Hellos/Goodbyes

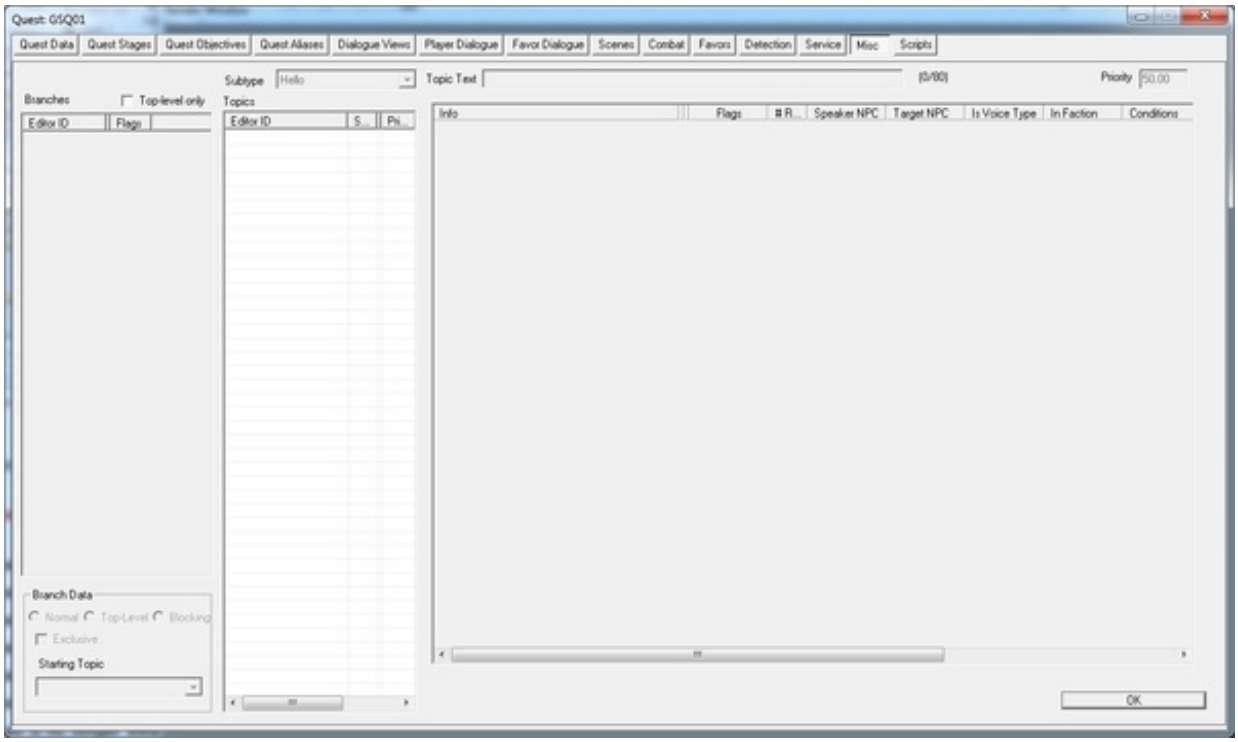
Right now when you interact with Bendu, he'll start with one of:

- "Yes, sera?"
- "And what might you need? Hmmm?"
- "Yes?"
- "Hmm?"
- "Need something?"

These are the generic lines written for the MaleDarkElf voice type, and serve as reasonable introductions to any conversation. But they are very bland and repetitive. Plus, if you plan on recording voices for your mod, they will sound different than the other lines you have (unless you plan to bring Keith Szarabajka in to record for you).

So we can write some specific lines for Bendu that will reference what we've done in the quest so far.

Open up the GSQ quest window, and go over the Misc tab.



Right click in the Topics table in the middle, and select "New." This opens up a list of special case topics that the game uses for specific cirumstances. Double-click on "Hello" to make a new topic of that type. When prompted to name it, go with "GSQ01Hellos".

This activates the table to the right, which is now functionally almost identical to the Info list we were dealing with when making dialogue. Only, in here, all these infos will be said by an actor when the player activates them. Just like with other infos, bad conditions in here can lead to everyone in the world having a new hello, so you need to be careful.

Right-click and make a new info. Something like "Oh, are you here to help?" Set its conditions to:

- GetIsID GSQBenduOlo == 1
- GetStage GSQ01 < 10

(If we wanted to, we could make a different hello for stage 5, but I'll leave that as an exercise.)

Make two more:

- "Have you got my amulet yet?"
  - GetIsID GSQBenduOlo == 1
  - GetStage GSQ01 >= 10
  - GetStage GSQ01 < 40
- "I can't thank you enough for helping me."
  - GetIsID GSQBenduOlo == 1
  - GetQuestCompleted GSQ01 == 1

That second line is actually not a great example, since it's asking a question that we haven't written prompts to allow the player to answer. But you get the idea.

You can also make a Goodbye topic, which will get used whenever the player leaves a conversation.

Note that the hellos and goodbyes, as written, will completely override the generic ones, and will be the **only** hellos or goodbyes that Bendu uses. We'll show how to mix things up a bit in the advanced tutorials.

Loose Ends

This should give you an idea of the kind of handling that needs to be done for every moving part of a quest. Even in this very simple example, there were lots of little odds and ends to attend to so the quest would be a more solid experience for the player.

By thinking this way, you can help make your own quests flow more smoothly and be more fun for the player.

Next Steps

This is the end of the Quest Design Fundamentals series. There are more advanced topics like radiant story that are discussed elsewhere in this wiki. If you want to dive more deeply into making quests for Skyrim, you should start with Packages, keeping Bendu's plugin still active.

Good luck out there!