Revision

Revision 1.0

Acknowledgments

Many members from the SWG Source Discord community assisted me in my research by answering questions, providing suggestions, and reviewing this document. In particular, I would like to thank Aconite[SM]#0069, Blood#4108, Cekis#7237, Bubba Joe#1963, Elour#2927, Erebus#1003, and Lord Horrar#5186. I hope I have not missed anyone.

Repository

This document and the sample quest are checked into HeronAlexandria's fork on GitHub. You can find my repositories by looking at the forks page from SWG-Source; Forks · SWG-Source/swg-main · GitHub or at HeronAlexandria · GitHub.

External Tools

I used Sytners to create and edit the string files and modify the client-side buildout file. For name generators, I use https://www.fantasynamegenerators.com.

The Simple Quest

Outside of the Mos Eisley starport, an Ansco Food employee waits. A group of thieves stole a prototype advanced shellfish harvesting tool, including the datapad with the schematic. The task is simple: go to a location, kill the thieves to get back the parts, collect the datapad, return to the employee and claim the reward.

Advanced Shellfish Harvesting Tool

The quest reward is a schematic for an advanced shellfish harvesting tool. The advanced tool has a use modifier, which affects the collection amount, and uses the harvesting character's survey and hand sampling skills.

There are three steps to creating the tool: defining the item, defining the crafting ingredients, and defining the schematic.

Creating the Tool

To create the tool, define the template, the shared template, and the script that allows the device to be used. I copied the existing files for the shellfish harvesting tool, renamed the files, and changed the copies.

The first file is adv_shellfish_harvester.tpf, located in swg-main/dsrc/sku.0/sys.server/compiled/game/object/tangible/harvesting. Change the shared template to point at the new shared template file and the scripts to use the script for the advanced tool.

The second file is shared_adv_shellfish_harvester.tpf, located in swg-main/dsrc/sku.0/sys.shared/compiled/game/object/tangible/harvesting. In this file, change the name and description. The name goes into "item_n" and "item_d". These are string files located on the client under string\en. I used Sytner's to edit these string files, but you may have to export these files from a tre file and then make your changes. You can save the files to the client directory, as demonstrated in my guide to creating vendors, so you do not have to place them in a tre file. You will find my string files in my client assets repository on Github.

The third file is the script that provides the functionality of the item. You can find the script adv_shellfish_harvesting.java, located in swg-main/dsrc/sku.0/sys.server/compiled/game/script/ /harvesting. This script is a modified copy of the shellfish harvesting tool. I will not detail this script, but I will highlight the changes.

Note that the calls to resource.createRandom include the player location, and these are prefixed with "seafood_mollusk" and "seafood_crustacean." Without these prefixes, the tool will randomly select shellfish from any planet in the resource tables. The prefix and player location restrict the shellfish to the character's current location.

The base chance to find shellfish is 20%, or the character's survey skill, whichever is higher. The tool's quality increases the likelihood of finding something and increases the chance of finding mollusks. The base chance is clamped at 95%, and the chance of finding mollusks is clamped at 50%. The character's survey skill and hand sampling skill increase the amount found. Feel free to change the script if you wish.

Once these changes are made, compile the tpf and script files. You will then need to copy shared_adv_shellfish_harvester.iff to the client (object/tangible/harvesting). Find the object_template_crc_string_table.iff for the client (it should be about 2 mb) and copy it to the misc folder in the client. You may then start your server and create and use the tool using administrator commands.

Creating the Crafting Template

The crafting template tells the crafting system what ingredients the schematic will use. The template consists of two files, one for the server and a shared file for the server and the client. These files are located in Located in dsrc/sku.0/sys.shared/compiled/game/object/draft_schematic/item and in dsrc/sku.0/sys.server/compiled/game/object/draft_schematic/item.

The names are a little confusing, so you might think that the files refer to an item, not crafting ingredients. The server-side file is item_adv_shellfish_harvester.tpf, and the shared file is shared_item_adv_shellfish_harvester.tpf. These files are simple, but I will go over them briefly. I copied the shellfish harvesting tool files and made the changes I wanted. In item_adv_shellfish_harvester.tpf I changed the locator array to an electronics GP module. Note the "ingredient type" variable. IT_resourceClass means a resource, IT_template means components from a crate (probably meaningful

if more than one is required), IT_templateGeneric means similar components (meaning that multiple items can be handcrafted).

Note that the quality of the tool is as high as 15, and the quality of the GP module is as high as 5. These quality values stack for quality of up to 20. Together, these make up the tool's use modifier. Since both of these characteristics are the use modifier, they will stack automatically.

The crafting template describes the ingredients for the crafting system but is not directly visible. To use the crafting template, we will need to define the schematic, the bridge between the crafting template and the actual object.

Creating the Crafting Schematic

The actual schematic is composed of two files, one for the server and one shared between the client and server. These files are located in dsrc/sku.0/sys.shared/compiled/game/object/tangible/loot/quest and dsrc/sku.0/sys.server/compiled/game/object/tangible/loot/quest.

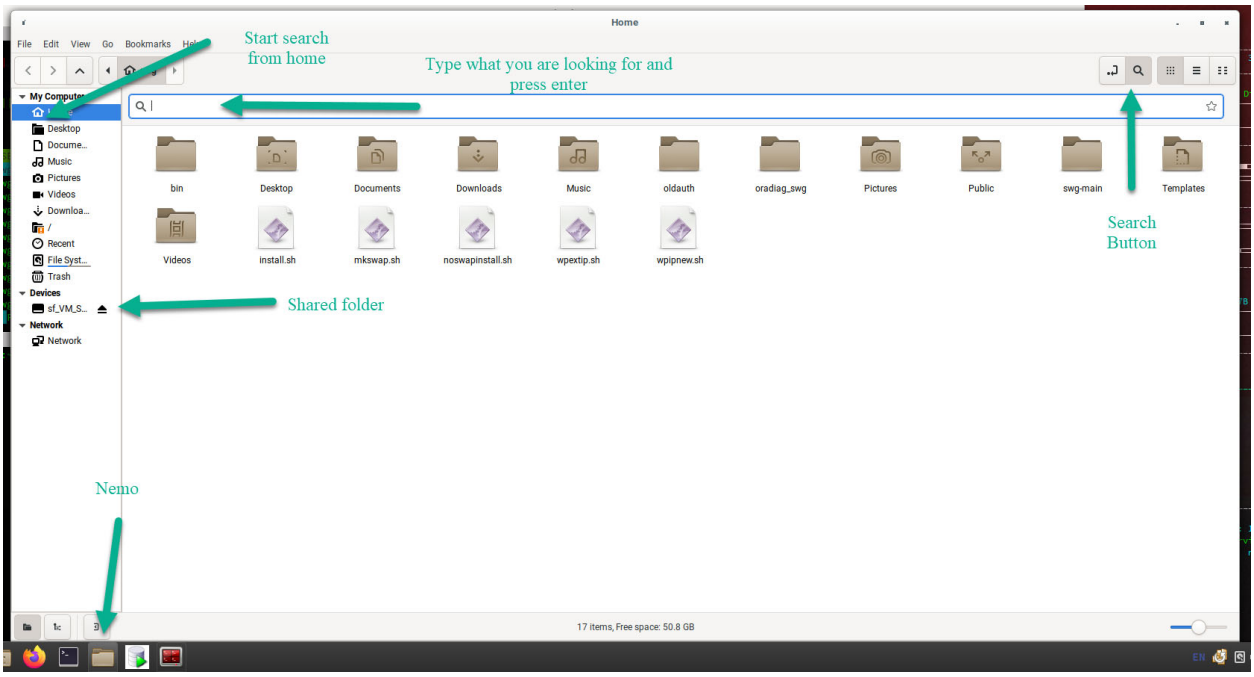
The server-side schematic file is adv_shellfish_harvester_schematic.tpf. It is a simple file, copied from the shellfish harvesting tool schematic and modified. You can use the sample file I checked into my fork or make your own. Note that the shellfish harvesting tool references were changed to the advanced shellfish harvesting tool.

The shared template is shared_adv_shellfish_harvester_schematic.tpf. This file contains the name and description of the tool stored in the string files and the appearance of the schematic. The string files are craft_item_ingredients_d and craft_item_ingredients_n. You may have to extract these from a tre file, and save them in the client's string\en directory. The only values you need to add are the names for the schematic; adv_shellfish_harvester_schematic. I have included my string files in my fork for reference.

Finishing the Schematic

Now recompile your server. Locate the two shared files on the VM that you have not copied, shared_item_adv_shellfish_harvester.iff, and shared_adv_shellfish_harvester_schematic.iff, and copy them to the client. The file shared_item_adv_shellfish_harvester.iff is to be placed in object/draft_schematic/item and shared_adv_shellfish_harvester_schematic.iff is to be placed in object/tangible/loot/quest.

The easiest way to search for files in the VM is to use Nemo, as illustrated in the following screenshot. You set up a shared folder on a hard drive for your VM, copy the files to the share, and then from the share to the client folder.



The last step is to find and copy the crc file to the client, as the schematic will be added. I would take a snapshot before actually testing anything. That way, if you made a mistake, it is easier to reset the VM to a state before you learn the schematic. You can make manual fixes when comfortable with how the server works.

You can now start your server, and you should be able to spawn the schematic, learn it, and craft the item. Once you have this done, it is an excellent time to create a snapshot of your VM.

Creating The NPC

Now we are going to create and spawn the NPC. The NPC will not do anything because we have to set up a conversation to interact with the NPC, but let's create the NPC and make sure it spawns correctly. We will set up the conversation later. I covered creating an NPC in my vendor guide, but I will briefly go over the procedure.

I used the Eisley human male guard as a base, converting the NPC into one that cannot be attacked. I used a Star Wars company name generator from https://www.fantasynamegenerators.com to generate a name. We are going to call the NPC "An Ansco Foods Employee." The completed NPC and general spawn location are shown in the screenshot below:

Use the following procedure to create the NPC:

1. Copy the template swg-main/dsrc/sku.0/sys.server/compiled/game/object/mobile; dressed_eisley_officer_human_male_01.tpf and copy to dressed_eisley_ansco_human_male_01.tpf
    a. Edit the file, and change the shared template to shared_dressed_eisley_ansco_human_male_01.iff
2. Go to the directory swg-main/dsrc/sku.0/sys.shared/compiled/game/object/mobile and copy the file shared_dressed_eisley_officer_human_male_01.tpf to dressed_eisley_ansco_human_male_01.tpf
    a. Edit the file
    b. Delete the name generator line. The name generator seems to override the static name.
    c. Delete the lookAtText line.
    d. Edit the objectName and change it to "human_ansco_male"
3. Find the creatures.tab file in dsrc/sku.0/sys.server/compiled/game/datatables/mob
    a. Copy an NPC, I copied clone_relics_mort and changed the name to ansco_male_01. I changed the planet to Tatooine (column J), the template (column N) to dressed_eisley_ansco_human_male_01.iff
    b. Clear the scripts (Column BN)
    c. Set can offer mission (Column BR) to zero
4. Add the spawner
    a. Refer to my vendor guide for details on adding a spawner and determining the coordinates and the buildout file to use.

b. I added the NPC to the tatooine_6_2.tab buildout file. For example, you can look at this file; the object id is -3000003.

5. Compile
6. Update the string names. This procedure is detailed in my vendor guide.
    a. Edit the creature names; mob/creature_names.stf. Add the string "human_ansco_male" and the value "An Ansco Food Employees."
    b. Copy the string name to the server as instructed in the vendor guide.
7. Copy the string crc to the client, as explained in a previous step.
8. Copy the shared file, shared_dressed_eisley_ansco_human_male_01.iff, to the client. The expected location is \object\mobile.
9. Start the server and make sure the NPC spawns.
10. Once you have this step working, take a snapshot.

Create The Thieves

Now we will add the thieves that stole the employee's prototype. I put them outside of Mos Eisely. Once we create the thieves, we will create a spawner. We are going to call the thieves "An Ansco Thief." The general location of the thieves and the datapad (we will discuss this in the next section) is shown in the image below.

Creating an enemy NPC is nearly the same as making the quest NPC. The only difference is that there are different settings in creatures.tab.

1. Copy dressed_eisley_ansco_human_male_01.tpf to dressed_ansco_thief_human_male_01.tpf
    a. Edit the file and change the shared path to shared_dressed_ansco_thief_human_male_01.iff.
2. Copy shared_dressed_eisley_ansco_human_male_01.tpf to shared_dressed_ansco_thief_human_male_01.tpf
    a. Edit the file and change the object name to "human_ansco_thief_male"
3. Add an entry to creature.tab. The provided settings make an NPC that is very easy to kill. Adjust the values if you wish.
    a. I copied the aakuan_anarchist, changed the name to ansco_thief_hm_01, and made modifications.
    b. Reduce the level to 5 (Column B).
    c. Set the planet to Tatooine (Column J)
    d. Set the social group to asnco_thug (Column K). The social group is very important; miss this, and the kill quest will not work correctly.
    e. Set the faction to bandit (Column L).
    f. Set the template to dressed_ansco_thief_hm_01.iff (Column N).
    g. Change the loot table to npc/npc_1_10 (Column AT)
    h. Change the loot list to generic_npc_1 (Column AU)
    i. Change the primary weapon to general_polearm (Column BT)
    j. Change secondary weapon to none (Column BV)
    k. Set aggressive to 0 (Column BZ)
    l. Set assists to 0 (Column CA)
    m. Set Deathblow to no (Column CD)
    n. I removed the special attacks.
4. Add the new NPC to the Tatooine_6_2.tab buildout file. Look for object -3000004 in the example I provided for an example.
5. Compile and do not forget to copy the share and the crc
6. Update creature_names and copy to the server; use the value human_ansco_thief_male and the description "An Ansco Thief."
7. Start the server, and the NPCs should spawn. In my example, up to four spawn at once.
8. Once you get this working, make a snapshot of your VM.

The Datapad

One of the tasks in the quest is to interact with an object. This object is a datapad on the ground near the thieves. This part is tricky because the server does not send static items to the client. Take a look at the image below. This is the datapad I spawned on my development VM. Note the zero in front of the object id. That means that this item comes from the client-side buildout file. We will talk about editing the client-side buildout below.

To create the datapad, perform the following steps:

1. Copy bounty_hunter_2_datapad.tpf located in dsrc/sku.0/sys.server/compiled/game/object/tangible/quest to ansco_stolen_datapad.tpf.
    a. Change the shared template to shared.ansco_stolen_datapad.iff
    b. Leave the script to pick up an item from the ground
2. Copy bounty_hunter_2_datapad.tpf located in dsrc/sku.0/sys.shared/compiled/game/object/tangible/quest to shared_ansco_stolen_datapad.tpf.
    a. Change the object name to ansco_stolen_datapad
    b. Change the detailed description to ansco_stolen_datapad
3. Add ansco_stolen_datapad to item_n; I called this an "Ansco Datapad."
4. Add ansco_stolen_datapad to item_d; I entered "A datapad stolen from Ansco Foods."
5. Position the item in the game world
    a. I started my server and went to where I wanted to place the datapad.
    b. I created the item and used /object rotate and /object move. I then used the buildout tool discussed in my vendor guide to build out files and coordinates.
    c. You can see those coordinates in my Tatooine_6_2.tab file, object ID -3000005.
    d. Add the datapad to the server buildout. Take a look at my buildout for an example. Note the script in column L; this is important because the quest task will not work correctly without it.
6. Compile
7. Add the item to the client-side buildout. I have included my buildout in my client assets as an example.
    a. To do this, you will need to extract the tatooine_6_2.iff buildout file from the tre and save it in the client directory, under datatables\buildout\tatooine. I used Sytner's for this task.
    b. Edit the file you just saved.
    c. Now we need to add a row for the datapad. Without this row, the datapad will not spawn on the client.
    d. We need the buildout id of the datapad (-300005 in this example), the shared_template_crc, and the buildout coordinates. Set the container to zero; the type is unsued; it can be any number. Use the buildout coordinates you got when you placed the item. In this case, the buildout coordinates are 1550, 5, 1597. You will notice that I rounded these numbers because Sytner rounds them, and I am not sure what will happen if they do not match the server's coordinates.

    e. To get the crc, find the object_template_crc_string_table.tab. I used the one in the /dsrc/sku.0/sys.client/build/game/misc directory. Open that file, and find the datapad; it will be called object/tangible/quest/shared_ansco_stolen_datapad.iff.
    f. Record the number after the 0x. That is the shared template id.
    g. I set the radius to 50, which is how close the player must be before the item spawns.
8. Copy the shared_ansco_stolen_datapad.iff to object\tangible\quest.
9. Copy the crc string file as you have done in the past.
10. Start the server, and then go to the location. You should see the datapad.
11. Once you have this working, take a snapshot.

The Quest

Now we have the pieces that we need. All that remains is to put them together. There are two significant components to a quest: the conversation with the NPC and the quest tables. The conversation is the most complex component and requires understanding how to create a simple state engine. Since this document is not a programming tutorial, I am not going over state engines. I hope that the example conversation I provided is sufficient to get you started, and I will discuss the basics.

In this quest, the conversation starts the quest and collects the reward. The actual quest is a series of tasks that are managed by the quest system. We will begin with the quest tables, as they are easier to follow, and it is essential to understand those so that the conversation makes sense.

The Quest Tables

There are two quest tables for each quest, one for the quest and one for the tasks that comprise the test. The data tables are shared and are placed in datatables\questlist\quest and datatables\questtask\quest. The quest's name is anso_recover_prototype, also the file name for both files.

To create this simple quest, follow this procedure:

1. In dsrc/sku.0/sys.shared/compiled/game/datatables/questlist/quest. Copy one of the files and rename it to ansco_recover_prototype.tab.
    a. This file is simple, so we are going to go over it briefly
    b. Set the level
    c. Set the tier. There are six tiers. This quest is a tier-one quest.
    d. Set the string descriptions, looking at the journal entries for the format. The strings are stored in the client folder, under string/en/quest/ground, in the file ansco_recover_prototype.stf. I have included my string file for this quest in my client assets repository, which I created using Sytner's.
    e. Set the experience points and experience type for the quest.
    f. Clear the reward columns if anything is in them.
2. In dsrc/sku.0/sys.shared/compiled/game/datatables/questtasks/quest. Copy one of the files and rename it to ansco_recover_prototype.tab. Depending on which file you copy, you may add some columns to the file. If one of the columns in the following steps is missing, you can add it and the datatype. Use the files I provided as examples.

a. Clear the tasks from the list. The tasks table is not complex, so I am only going over it in detail. Look over the entries; they are generally easy to follow, but I will comment on the less obvious aspects.
b. The tasks are to go to a location, kill some NPCs, pick up the datapad, go back to the NPC, and talk to the NPC for the reward.
c. Note that the visible flag on the tasks is set, and note the TASKS_ON_COMPLETE. This determines what tasks become active as the task is completed.
d. I have named each task, although this is only necessary if you need to check the task in a script.
e. CREATE_WAYPOINT must be 1 to create a waypoint on the given planet at the provided coordinates. Note that these are not buildout coordinates but world coordinates. If the waypoint exists in the character's datapad, it will not be created.
f. Waypoint names are taken out of the quest string file; refer to column AB in the example data table.
g. The first task is to go to a location. The radius is how close the player must get before the task completes and advances to the tasks defined in TASKS_ON_COMPLETE.
h. The second task is to kill the thieves until three parts are collected.
    i. Kill tasks target a social group, defined in column AD, SOCIAL_GROUP. Remember when I mentioned the social group when defining the thieves? The two names must match.
    ii. The name of the loot item is defined in LOOT_ITEM_NAME (column AI). The number of items required to complete the task is defined in LOOT_ITEMS_REQUIRED, and LOOT_DROP_PERCENT determines the probability of a drop per kill. No items drop; you will see an update as you collect items.
i. The third task is to collect the datapad from the ground.
    i. Make sure you set the drop percent to 100 (DROP_PERCENT, column AL), or the task will not complete.
    ii. Set the text for the retrieved menu in RETRIEVE_MENU_TEXT, column AM.
    iii. Set the server template for the item. You will see this in column AF, SERVER_TEMPLATE. This is set to the server's stolen datapad. When the player interacts with the item on the client, the server will map the item on the client to the item on the server. Remember, the datapad is not sent to the client by the server.
    iv. Set the number required to one, in NUM_REQUIRED, column AG.
    v. The item name is set in column AH, ITEM_NAME.
j. The fourth task is to return to the NPC. I did not set a radius for this, so if you have trouble completing the tasks, you might try setting the radius. I did not have any trouble with the radius empty.
k. The fifth task is to wait for a signal named collect_reward, defined in SIGNAL_NAME, column AP. The NPC sends that signal when the player responds that the quest is completed, which completes the task.
l. The sixth task completes the quest, which happens when the preceding task completes.

3. Compile.

4. Copy the ansco_recover_prototype.iff files to the client, placing them in datatables\questlist\quest and datatables\questtask\quest. Do not get the two tables mixed up.
5. Copy quest_crc_string_table.iff from the server, located in /home/swg/swg-main/data/sku.0/sys.shared/built/game/misc to the misc client directory. This is similar to the crc file you copied for objects, but contains the crc for the quests.

The Conversation

Conceptually a conversation is very simple. It is a java file that controls what the NPC says to the player, what response the player can make to the NPC, and how the NPC responds. Conversations are thus script files stored only on the server in the script/conversation directory.

I created ansco_employee_eisley.java by copying the clone armor quest and modifying it. You are welcome to use my script as a base. Remember that you must change the class name to match the file, and there are functions in the original script that are not required for this simple quest. The original conversation was generated with a tool, making it hard to follow.

When a player talks to an NPC, the server calls the method OnStartNpcConversation in the conversation script for the NPC. This method is used to decide how to respond to the player. When the player first interacts with the NPC, there are two basic options: have the NPC say something simple to the player, or set up responses the player can make and start a conversation.

To have an NPC say something to the player, call npcSpeak, start a conversation, and call npcStartConversation. For example, if the NPC offers a quest, you would set up the character's responses to accept or refuse the quest and start a conversation. If the player has completed the quest, the NPC can thank the player.

When a player selects a response, the server calls the method OnNpcConversationResponse. The call includes the player's response to make decisions based on the selection. For example, if the player accepts the quest, the NPC can thank you and tell the quest system to grant the quest.

Conversations do not have to be that simple. In the sample quest, the NPC begins by complaining, and the player can ask what is wrong. In this case, the NPC describes the problem, sets up the accept or refuse responses, and continues the conversation by calling npcSetConversationResponses. To end the conversation and say something to the player, call npcEndConversationWithMessage.

So, to say simple to a player, call npcSpeak, start a conversation call npcStartConversation, and end a conversation with a message call npcEndConversationWithMessage. There is also npcEndConversation, which ends a conversation without a message, but I do not use it in the example script.

Now, let's go over ansco_employee_eisley.java, the conversation script I wrote for this example. When the player talks to the NPC, the first thing we do in OnStartNpcConversation is to check and see if the character has the quest. There are two possible states if the character has the quest. Either the player is on the collect reward task or is not.

If the player is on the collect reward task, we set up the responses in the method ansco_employee_eisley_NPC_SetupCompleteQuest to tell the NPC that the quest has been completed.

Note that we set a script variable on the player to track where we are in a conversation. This script variable contains the "branch" of our conversation. Note that in this case, we call npcStartConversation because we are starting a conversation. Just examine the script to see how this script variable is managed.

You will also note that we check to see if the player is a GM. I have set up this quest to allow GMs to reset (clear) the quest or ask for the reward. For this reason, a GM will have responses were regular players will not. You can use this technique to test your quests.

If the player has the quest but has not completed the quest and is not on the collect reward task, the NPC simply speaks to the player, saying to complete the quest. This is accomplished by calling npcSpeak. You can do animations; refer to the script for calls to doAnimationAction.

If the player has completed the quest, the NPC thanks the player. If the player is a GM, then the player gets the GM responses, which allows the player to clear the quest or claim the reward.

If the player does not have the quest active and has not completed it, the NPC sets up the responses that allow the player to ask about the problem.

Note that the above narrative describes setting up the responses, so we will now decide how we process the responses for the player. When the player selects a reply, the server calls the method OnNpcConversationResponse in the conversation script.

Looking at OnNpcConversationResponse in the sample script, you can see that the conversation branch acts upon the responses. I will not cover the response logic in great detail, but I will make a few general comments, so you get the idea.

If the NPC is complaining about his problem, we set up a response that allows the character to ask about the situation. If the character asks about the problem, we set up the answers to enable the player to accept or refuse the quest. Note that is an example of continuing a conversation.

If the player accepts the quest, then the script grants the quest. If the player refuses the quest, the NPC makes a snarky response, and the conversation ends.

If the player tells the NPC that the quest has been completed, then the script signals the quest system to complete the task, which completes the quest. The NPC complains about the broken prototype but gives the character the schematic.

To converse with the NPC, make sure you set the CONDITION_CONVERSABLE object variable and show the quest icon to set the CONDITION_INTERESTING object variable. Examine the provided conversation for examples on how to use these object variables.

Enable the Conversation

Now that we have the pieces in place, we need to finish up the entry in creatures.tab. We are going to update the entry named ansco_male_01. Place the script name conversation.ansco_employee_eisley in the column (column BN). Set canOfferMission, column BR, to 1.

Once you have made these changes, recompile, and your quest is completed. Make a snapshot, and test your quest.

Animations

These are the animation names that I found while researching this guide.

2hot4u	accept_affection	action	adjust
airguitar	alert	angry	apologize
applause_excited	applause_polite	ashamed	backhand
backhand_threaten	bang	beckon	belly_laugh
blame	bounce	bow	bow2
bow3	bow4	bow5	catchbreath
celebrate	celebrate1	check_wrist_device	cheer
chicken	chopped_liver	clap_rousing	claw
clientAnimation	conversation_1	cough_heavy	cough_polite
cover_ears_mocking	cover_eyes	cover_mouth	cuckoo
curtsey	curtsey1	dismiss	dream
eat	elbow	embarrassed	emt_nod_head_once
emt_stand_confused	expect_tip	explain	face_eye_roll
face_innocent	face_wink	fakepunch	flex_biceps
flex3	flipcoin	forage	force_choke
gesticulate_widly	gesticulate_wildly	giveup	goodbye
greet	hair_flip	hands_above_head	hands_behind_head
handshake_tandem	he_dies	heavy_cough_vomit	helpme
hi5_tandem	hold_nose	hug_self	hug_tandem
huge	huh	implore	kiss
kiss_blow_kiss	kisscheek	laugh	laugh_cackle
laugh_pointing	laugh_titter	listen	look_casual
look_left	loser	manipulate_high	manipulate_low
manipulate_medium	medium	mistake	mock
nervous	nod	nod_head_multiple	nod_head_once
offer_affection	paper	pat	pet_creature_medium
pet_high	petAnim	point_accusingly	point_away
point_down	point_forward	point_left	point_right
point_to_self	point_up	poke	pose_proudly
pound_fist_chest	pound_fist_palm	refuse_offer_affection	reload
rofl	rose	rub_belly	rub_chin_thoughtful
rude	salute	salute1	salute2
scare	scared	scratch_head	scream
search	shake_head_disgust	shake_head_no	shakefist
shiver	shoo	shrug_hands	shrug_shoulders
shush	sigh_deeply	sit_trick_1	sit_trick_2
slit_throat	slow_down	slump_head	smack_self

small	smell_air	smell_armpit	snap_finger1
snap_finger2	sneeze	spit_hands	squirm
stamp_feet	standing_placate	standing_raise_fist	std_manipulate_medium
stop	stretch	strut	survey
sweat	taken_aback	tap_foot	tap_head
taunt2	thank	threaten	threaten_combat
throwat	thumb_down	thumb_up	thumbs_down
thumbs_up	tiny	tiphat	trick_1
trick_2	trickName	twitch	udaman
vocalize	waft	wave_finger_warning	wave_hail
wave_on_directing	wave_on_dismissing	wave1	wave2
weeping	whisper	worship	wtf
yawn	yes		