

TES Construction Set: NPC Tutorial, Part 1

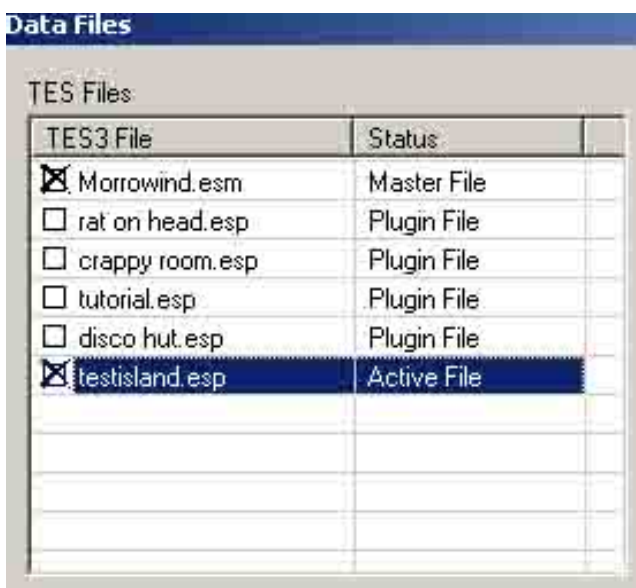
Welcome back to the TES Construction Set tutorial. In the last installment I gave some helpful information about finding your way around the construction set, using the controls, loading files, editing indoor spaces, landscaping outdoor spaces, connecting doors together through the teleport properties and a lot of miscellaneous tips. It should have given a fairly good idea of the things you can do with the construction set and gotten you used to the basic controls.

In this tutorial, I'll try to cover many aspects of NPCs (non-player characters) so that you can add characters to the world, change aspects of their appearance and personality and add some dialog and scripting to them. Although NPCs are a little complicated, they're a vital part of designing a good game and good quests.

When working through the NPC tutorial, I'll begin from a simple quest scenario that illustrates a typical case.

Marius, the lighthouse keeper on the island has not been feeling well lately because he keeps getting diseased from various critters crawling around the island. He could really use your help in getting some cure potions that he knows his friend Cornelius has. The problem is that he can't leave the lighthouse to go fetch the potions because he needs to make sure the fire stays lit so that ships don't crash on the rocky shoals. If you accept the quest, he'll give you 200 gold pieces and ask you to trot down the hill on the island, give the gold to Cornelius in return for the 3 potions, and trot back up the hill to bring them to him. In return, he promises to give you a "useful item" for your work (a silver longsword).

To discourage foul play (or sneaking and stealing) Marius has some cursed longswords around. If you complete the quest, he gives you the non-cursed variety. If you simply kill him or steal the sword lying around his house you're in for some extra surprises.



Open the file you wish to modify

I'm going to walk through this example by adding some items, a couple of NPCs and a cave to the test island I made for the the last tutorial. If you followed along for the last tutorial you may have these items, but feel free to download the [test island](#) that I've put together that gives a basic starting point.

After copying the testisland.esp file to the \Program Files\Bethesda Softworks\Morrowind\Data Files folder, it should show up in the dialog for loading the datafiles into the TES editor.

Load these files and make sure testisland.esp is the active file.

Go to cell -10, -5 to view the island and the lighthouse

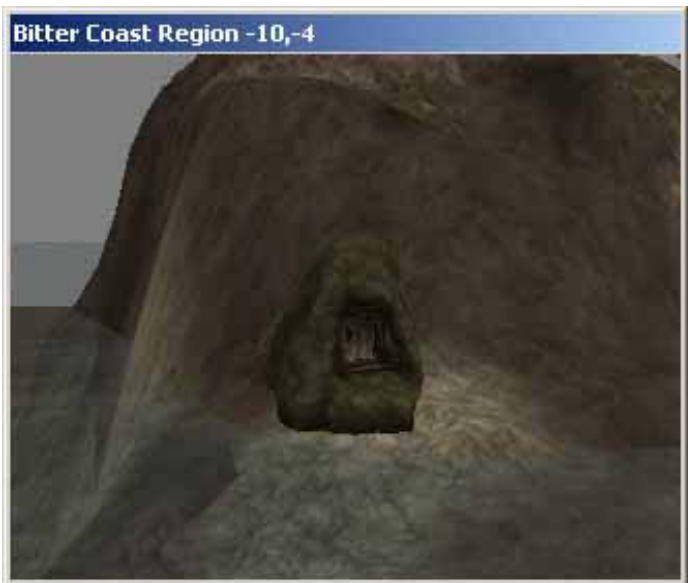
Try World | Go to cell ... as an easy way to get to this cell quickly without having to scroll through the whole cells list-- simply type -10, -5 into the dialog and it will take you there (unless you're currently editing an interior cell).

Add a cave entrance, door and cave interior cell



You'll now need to make a small cave for Cornelius to live in. I've added a cave entrance to the hill (ex_bc_cave_entrance) and a door (door_cavern_doors10).

I've also added an interior cell to match the exterior and called it "Cave of Cornelius." The pieces I used were of the in_mudcave_2_s variety which make up a small cave. I used the doorway in_mudcave_doorway00 and a fairly standard cavern door (door_cavern_doors10). Be sure to set the teleport properties of the door so that both the interior and the exterior doors work correctly and take the player to the right places.



I set a lock on the exterior door (at level 50) because I'll have Marius give the player a key to open the door. The lock information is set in the same properties window where the teleport information is set for the door. I'll make the key a little later.

If you want to tour your setup in Morrowind to make sure it all works so far, go ahead. Use the command `coe -10, -5` to teleport yourself to the island and you may unlock the door without using your skills by entering command mode (press ~), clicking on the locked door, and typing `unlock`.

Add Cornelius as a new NPC type

Click the NPC tab in the object window, right click inside the object window and choose New.

I'm going to call the character "Cornelius Carpaetin" and set some properties for him as are shown in the pictures on the left. (I cut this dialog in two to keep the width manageable.)

Although I've set the ID and the name the same, they don't

NPC

ID

Name

Script ...

Race Female ☐

Class Level

Faction & Rank

Essential ☐ Corpses Persist ☐ Respawn ☐

Attributes

Str	79	Spd	43	Health	113
Int	36	End	68	Magicka	72
Wil	58	Per	34	Fatigue	253
Agi	52	Luc	40	Disp	50
				Rep	0

Blood Texture ☒ Auto Calculate Stats

Value	Skill
22	Restoration
6	Conjuration
6	Destruction
6	Enchant
50	Axe
14	Long Blade
46	Blunt Weapon

Dialogue Animation AI

have to be this way. The ID is just a reference the program uses to track the NPC object, while the name is what the player will see displayed. It's probably smart to use the same name and ID when possible, though.

All the information in the dialog for stats can be filled in and will probably seem familiar since the non-player characters have the same attributes that your own character has. Notice the **Auto Calculate Stats** checkbox which automatically fills in stats based on the character's race, class, level and other information you supply. Checking this box is a convenience so that you do not have to fill in stats for every character you create, but if you wish, you can uncheck the box and customize the stats to your liking. The Auto Calculate box must be unchecked in order to use some of the AI settings to allow an NPC to buy or sell items or perform some special functions.

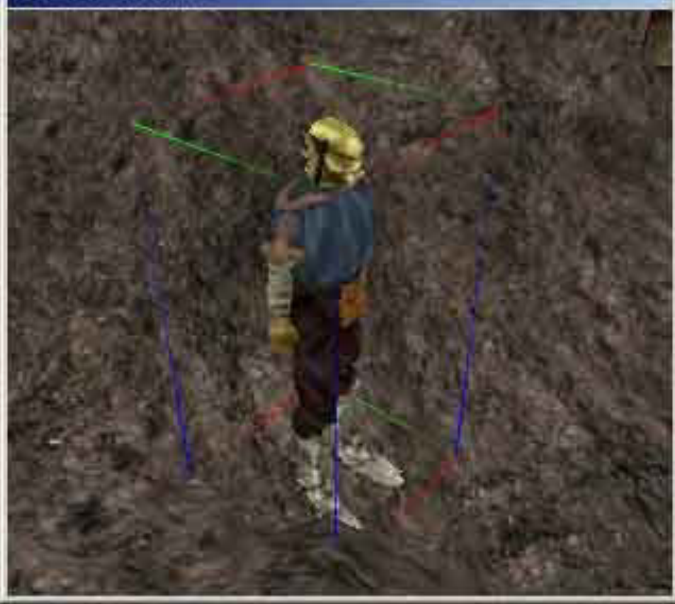
I'll get to the script box later as I show the process of scripting for NPCs and for weapons we create, but for now just leave this as it is with no script specified.

In the right half of the NPC dialog (bottom of the two pictures) you'll see that you can choose the head and hair for the character. It's too bad you can't see these items displayed immediately when they are selected, but you'll get a chance to see what the NPC looks like when we drop him into the world.

Towards the bottom of the dialog you'll notice that you can give the NPC items and spells—although you can give these to the character right now, we'll wait on these items for when we edit the information from the Render Window.

Note: At first I thought that defining an NPC in this dialog was like defining a programming class and that dropping the NPC in the world itself was like creating an instance of the class (which starts from the class template but can be modified independently). The object in the render window is not an instance of the class as you might expect—for example, if you add items to the inventory from the render window's properties dialog, then these items show up under the properties window accessible from the object window also. Both share most of the same properties. If you have more than one of a character object in the world (such as for guards) then these duplicates become instances of their character type at run time. If you didn't understand a word of this paragraph, don't worry—you just need to know that whatever you change for a NPC from either the object or the render window changes the other (except for world position data and a few other items).

Cave of Cornelius



some clothing you would like him to wear and drop it into the inventory (items) in the Cornelius' properties window. After you've dropped all the clothing in his inventory, click the save button to save the changes and you will see what he looks like with the clothing you've selected for him. You may also change the head and hair type in the properties window. You will not see the changes until you click Save.

If you do not like the way things look, enter the properties window again, delete the items you do not like and try adding other items. You may also add pieces of armor or weapons from other categories besides clothing. It would be useful to give characters some weapons and armor so they can defend themselves against attack.

I've looked at the weapon skills on Cornelius and given him an axe since he seems to be good at using one.

Add Marius as a new NPC, put him into the lighthouse and give him clothing

Add Marius as another NPC in the object list. Enter the properties you see at the left or customize the properties to your own liking.

After Marius is added to the object list, find the cell (in the Cells Window called "Abandoned Lighthouse" and change the name to "Remote Lighthouse." After all, it will not be an abandoned lighthouse anymore once Marius has moved in and is keeping the beacon alight.

Now move Marius into the the lighthouse and give him some clothing as explained in the previous step. You may also change his head and hair to suit your taste—go ahead and experiment. Give him a weapon and at least some armor.

If you were to test your plugin at this point you would find that these characters are boring since all they want to do is stand around (sometimes wandering) and tell you the kinds of things you could hear from most people you see wandering around any town. Also there isn't much to see in their houses (the cave and the lighthouse).

Add clutter and appropriate items to the lighthouse and cave

Add some items to both the lighthouse and the cave to make these locations look lived in and more realistic. I've added a bed or bedroll, some barrels and other items. Adding items should be nothing new—only this time we're going to give the items ownership by the NPCs in these locations.

NPC

ID:

Name:

Script:

Race: Female ☐

Class: Level:

Faction & Rank:

Essential ☐ Corpses Persist ☐ Respawn ☐

Attributes:

Str	46	Spd	54	Health	64
Int	60	End	43	Magicka	120
Wil	34	Per	71	Fatigue	160
Agi	37	Luc	40	Disp	50
				Rep	4

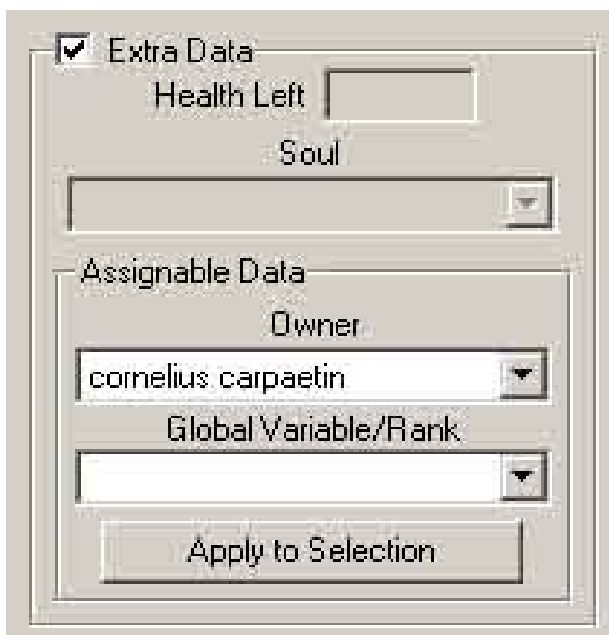
Blood Texture: ☒ Auto Calculate Stats



Set ownership for items in each NPC's house

Go into one of the NPC's areas and double click on an item that you've just added. Check the Extra Data box and assign Cornelius (or Marius as appropriate) as the owner of the item from the Owner drop-down list. This will assign the ownership for this one item.

To easily assign ownership for multiple items, simply select the items from the objects list on the right side of the cells window and then click on the Apply to Selection All items that were selected will be assigned to the current owner that is shown. To select multiple items from the list, press the CTRL key while clicking each one.



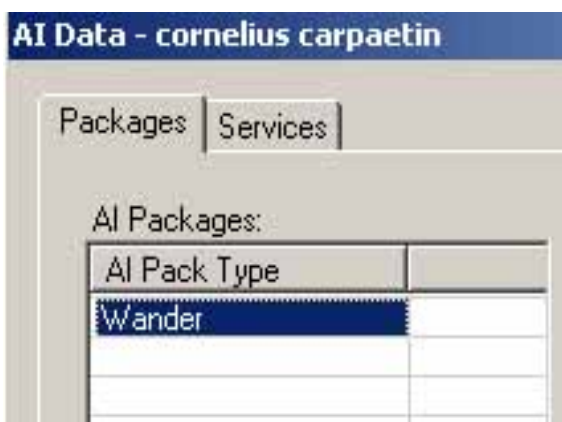
Why do you want to give ownership?

By giving ownership, the player cannot take items without it being considered stealing. Since these two characters seem to be living in these locations, it makes sense to assign ownership to the items here so that the player cannot simply take them with no consequences.

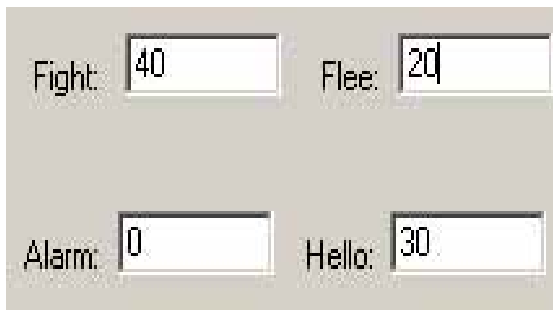


Make them stop walking around and change the fight or flee settings

If you've checked on your NPCs by loading the game and looking at them, you've probably noticed that they are wandering around in a stupid way. Marius particularly seems to wander around the lighthouse in a dumb way. He keeps walking on the edges of the room and walking on the bed and other things. Lets make him stop.



Open up your character by double-clicking him in the render window, click the AI button and then remove the Wander AI package by pressing the delete key on the keyboard while it



A screenshot of a game's NPC configuration window. It features four input fields arranged in a 2x2 grid. The top row has 'Fight:' followed by a box containing '40' and 'Flee:' followed by a box containing '20'. The bottom row has 'Alarm:' followed by a box containing '0' and 'Hello:' followed by a box containing '30'. The background is a light beige color.

Property	Value
Fight	40
Flee	20
Alarm	0
Hello	30



While we're here, I'm going to make Cornelius a little more prone to fight and less prone to flee, since he is an orc after all. Yes, maybe Tamriel has misunderstood orcs who are not really so bad at heart—but I expect that orcs are more likely to fight than other races.

I'm moving his fight rating up to 40 and his flee rating down to 20. Essentially each of these areas (fight, flee, alarm) have a rating in the range of 0 to 100 where 0 means the character is not very likely to take the action and 100 means they will take the action almost every time. Factors such as the NPCs disposition toward the player come into the formula for when they will fight, flee or raise an alarm.

The Hello box sets the distance from you at which the NPC will say hello. 30 is the default and I'll leave it alone.

Save your changes, test and make any modifications needed

Save your changes to the plug-in file, open Morrowind and test the changes and come back to make any other changes to make things work better.

The next part will explain how to give these NPCs some useful dialog besides the current dialog that has been assigned to them by default because of their class, rank, race, etc.

TES Construction Set: NPC Tutorial, Part 2

Now that you've created some new NPCs and set them up, the next part of this tutorial will involve understanding the Dialogue Window and adding some journal entries. If you'd like you may [download the plugin](#) that I've created up to this point.

Since the dialog function becomes complicated, I'll spend a fair amount of time explaining the basic functionality for the dialog window. Don't be too overwhelmed by the dialog window since you can figure it out by taking things a piece at a time.



Get to the dialog information for Marius Obertam

Double-click Marius to get the properties sheet and click the Dialogue button to work with dialog.

Note: The dialog box may take a few seconds to open since there is a lot of dialog in the game and the editor loads all of it and filters it for you.

Dialogue Window Overview

The dialog window is one of the most complicated windows in the construction set and so I'll have to explain the details of each section piece by piece. It's a good idea to get an overview from the picture at the left.

The Tabs section allows you to see one type of dialog at a time—either topic, voice, greeting, persuasion or journal information.

The Topics section gives you a listing of the topics that are available (or the categories that are available on the selected tab). You may not see everything that is available if you have the filter for set to only show topics available for a specific character. Even fewer topics may be available to the player when playing the game than are shown with the filter for option turned on. I'll explain more below.

The Info / Response section shows a summary of the items that are available inside a topic. The Info / Response seems wrongly named to me since it is really just an item summary listing for the currently selected topic.

The Details about selected item is not named in the construction set—but is labeled as three different sections (see the tan box in the picture). Since these three sections all give in-depth information about the selected item from the Info / Response section, they function as one properties window to edit and view properties for the selected item.

The Troubleshooting section gives you tools for getting additional information about dialog items and for taking certain actions.



Using the dialog window to find or edit information

1. Select the dialog type you wish to find (from the tabs), and filter by a character if possible to make the topics list a manageable size.
2. Select the topic from the left side of the window. Although the type of items you're working with depends on the tab selected at the top (some are greetings, voices, persuasions or journal entries). I'm going to use the term **topic** to describe all kinds of information you work with on this left side. These entries function like categories (or topics) no matter which tab is selected.
3. Select an item in the Info / Response area to open the properties for the item in the bottom part of the window.
4. View and edit the full properties for the item in the bottom part of the window—the properties you may edit include the dialog text, the speaker conditions and the results (scripting) for the item.



Selecting an item type and filtering the list

Select a tab at the top of the window—they include:

Topics—the things NPCs may talk about.

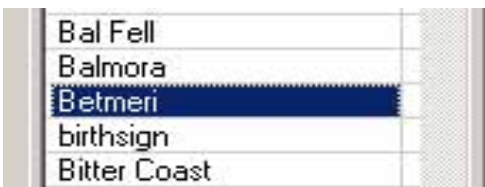
Voices—the sounds NPCs make at various times.

Greetings—the things NPCs may say when the player first begins talking to them.

Persuasions—the ways NPCs may react to the player's attempts to persuade them.

Journal—the entries that may appear in the player's journal.

After selecting a tab, you may choose to filter for the current character you are editing so the topic list isn't so huge and unmanageable. By default, when entering the Dialogue Window from the NPC editing screen, it automatically filters by the NPC you're working on—so you will only see the topic and item lists that this character might be able to talk about.



Select a topic

When you select a topic from left side you will see the items in the Info / Response section that fall under that topic. The topic list will be filtered according to your filter for settings.

Select an item from info / response

Click on an item in info / responselist to display or edit

Info / Response	
Info	Disp/Inc
The peaceful Khajiit and Argonian races are the war-loving Orc tribes are relatively few in number. Other smaller Beast races, like Goblins north of Tamriel, and seldom encountered in the	30
Betmeri, or 'Beastmen,' were th...	30

The peaceful Khajiit and Argonian races are the war-loving Orc tribes are relatively few in number. Other smaller Beast races, like Goblins north of Tamriel, and seldom encountered in the

Speaker Condition

ID

Race

Class Savant

Faction

Rank

Cell

PC Faction

PC Rank

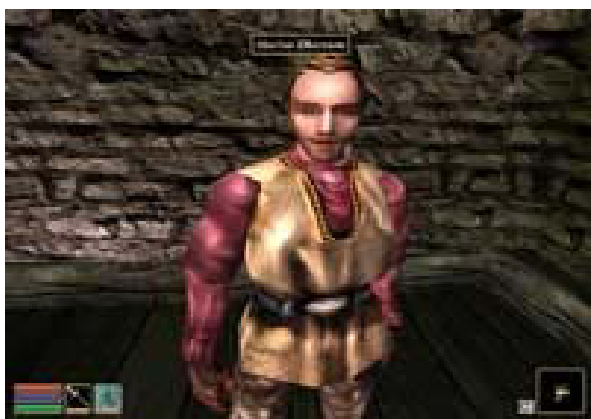
Result

; bard, bookseller, savant;; CHOICE 1;

Sex Disp 30

Function/Variable

Function	Choice	=	1
Not Local	NoLore	=	0
		=	
		=	
		=	
		=	



information about it in the lower half of the screen.

Although the grid layout shows you some information about the item, it is only a brief summary of the item and you'll need to look at the properties section below to get all the information.

Edit or view the information in the details section

The top of these three pictures shows the dialog section—where you can view, edit or enter the dialog you would like to appear when the dialog item is "spoken" to the player.

The second and third of the pictures shows the conditional area. It specifies the conditions that must exist for the non-player character to give the dialog that is shown in the dialog box above it.

The results section (very bottom of the second picture) allows simple scripting actions that will occur if the player receives the current dialog item from the NPC during the game.

The Speaker Condition section is the most complicated in designing your dialog (except for the results section—which we'll talk about with the scripting). If you want to limit a piece of dialog so that the player only gets it when talking to a specific NPC or an NPC of a specific race, class, faction or rank, set the speaker conditions here. The decision-making process that directs an NPC to deliver dialog to the player can get very specific to a character or situation since it is based on the items in the speaker condition section.

Rules for how the dialog works in Morrowind

Even if you understand how the Dialogue Window is set up there is still a lot to understand about the way Morrowind dishes out dialog from NPCs. Going through an example will probably explain a lot. For the example, take a look at the last four pictures above. They show a piece of dialog about the Betmeri in the game.

According to the Dialogue Window in the construction set, Marius Obertam has some dialog about the Betmeri—yet when I go talk



to him Betmeri does not appear in his list of topics (see pictures at left). Why is this?

Rule #1: If the player has never heard of a topic, it will not be a choice when talking with an NPC. Apparently my character in the game has never heard of the Betmeri so it doesn't appear in the list of topics. A player hears of a topic when the topic word appears in any dialog he reads (either from the same NPC or another one). For instance, if my player had talked to someone in Balmora and the person said, "I hate the Betmeri. They are always playing their music so loud, those whippersnappers," then I would have heard the topic word Betmeri already; the topic then might be available to my character when talking with Marius (depending on the other rules, also). If there were a topic for playing their music so loud or for whippersnappers then I would have heard of these topics already, also. If you need to introduce a keyword to a player so that he can ask more about the topic you need to include the keyword in a greeting that the NPC makes or inside an item in another topic. The dialog linking in the game occurs automatically so you don't need to provide hyperlinks. It figures out the keywords from the topics list.

Rule #2: Dialog must satisfy all speaker conditions to be delivered to the player. This means that ALL conditions that are filled in under the speaker condition section must be true for the player to read the dialog. Even if my character had heard of the Betmeri already (rule #1) it still doesn't necessarily mean that I will get to hear this dialog item about the Betmeri from Marius when I'm playing the game. If you look at the speaker conditions shown in the screen shots up the page, then the following conditions need to exist for my player to get the dialog from Marius: the NPC must be of the class Savant (which he is), I had to choose choice #1 from a previous dialog (which may or may not be true), Marius' disposition toward me must be higher than 30 (which may or may not be true) and the variable NoLore must not exist in the local script (which may or may not be true). As you can see, many conditions cannot be calculated until Morrowind is running since they depend on variables and items that will change depending on what the player has done in the world and on the player's attributes. Items that cannot be calculated until the game is in play are known as run-time properties (since they cannot be figured until things are run).

Rule #3: The first line-item in the Info / Response section that evaluates as true will be used and all other line items after it will be ignored. What does this mean exactly? Notice in the Info / Response item section in the screen shot above that there are two items in the list? If the first item has its conditions met (rule #2) then it will be delivered to the player and all other items after it will be ignored—even if the items after it also evaluate to true.

The Morrowind engine evaluates the items in the list in the order in which they are listed and stops evaluating the list once it finds an item that is true. This means that the order of the items in the Info / Response section is very important—you generally want to

have the items with the most restrictive conditions (according to rule #2) at the top of the list and the items with the most relaxed conditions at the bottom of the list. By putting restrictive conditions towards the top of the list and relaxed conditions towards the bottom you ensure that the relaxed conditions don't hijack the the restrictive conditions from ever being evaluated.

For instance, if I assigned two pieces of dialog to Marius about the same topic and he was supposed to say one only when there was a full moon and the other any time (without restrictions), then I would want to put the "full moon" piece of dialog first in the list. If I put the non-restricted piece of dialog first in the list, then it would be checked first to see if it was true—and it would be delivered even when the moon was full and I really wanted the "full moon" dialog to be delivered instead.

Remember, the list stops being evaluated when the first true item has been found. Because the order is important you may re-order items in the Info / Responselist by pressing the left and right arrow keys on your keyboard.

Writing the dialog for the character interactions

Ok, enough theory for right now. It's time to add some items to the Dialogue Window [Download the dialog](#) so you don't have to type it all in by hand. If you are writing your own dialog, I'd recommend doing some rough documentation showing how things will work and the topic words you'll use so you don't get confused while working.

Read over the dialog sheet to get the idea of how all the interactions will take place. If you don't understand everything, don't worry since I explain it in more depth later.



The dialog file I've supplied is in rich text format which you can open with any word processor or even with WordPad (which comes as part of Windows). Instead of typing everything in you can just select dialog and copy and paste it into the correct places as you build this scenario.

Open the Dialogue Window and enter a journal to pic (category)



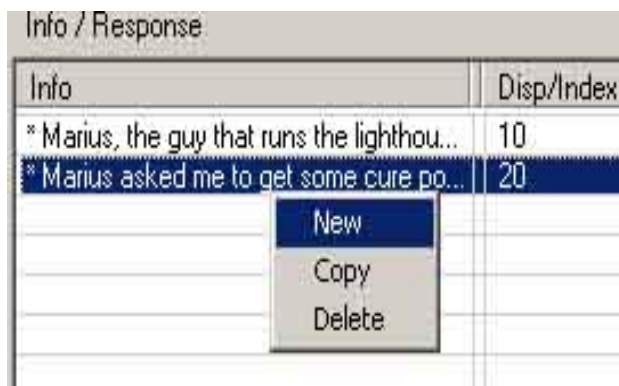
Click on the Journal tab (if you can't see it, scroll the tabs sideways with the arrows). Now, right click anywhere in the topic section and click New. Name the topic XZ_Marius .

If you cannot see the new topic you just created, do not create it again—change the Filter for section so that you are not filtering for a NPC. Click on the topic you've just created in the journal, XZ_Marius .

If you're wondering why many topics have a two-letter prefix in the journal, there are a couple of reasons: (1) Most topics with the same two letter prefix are related and (2) entering words without

an unusual prefix gives unexpected results since the engine will deliver entries from the the journal if they come up in conversation. The game will use journal topics like regular topic keywords (the characters can "speak" journal entries if they mention the journal entry topic word in their speech beforehand).

For example, I began by making the journal entry category Mari us instead of XZ_Marius . When I talked to Cornelius, he said the word Mari us in one of his speeches—Mari us was then underlined like any other topic word, so I clicked on it. He then spouted out a journal entry as a speech (which didn't make much sense). Once I changed the journal category name to XZ_Marius , the word Mari us would no longer match XZ_Marius . The word Mari us was no longer underlined since it wasn't an exact match to the journal entry topic and I stopped getting random journal entries "spoken" to me by other characters that were not supposed to speak them. If you don't understand this complete explanation, take my word for it, you do not want to name your journal categories for words that will come up in norm al conversation in the game.



Put items #1 and #2 in the jour nal topic you just created

Make sure XZ_Marius is selected on the left and then right click in the Info / Responsesection and choose New. Enter the entry text in the text box below the Info / Responsesection. It's safer to type (or paste) text into the full-size box since entering text in the info list (like you're changing a file name) cuts off your typing after a very limited number of characters.

Ensure that you enter the index number for each journal entry since you will need to reference these numbers later in a script.

Ok, that's it for part 2 of the NPC tutorial. A lot of part 2 was conceptual since it's important to understand the workings of the Dialogue Windowand Morrowind to design dialog and quests so they work correctly. [Part 3 of the NPC Tutorial](#) will probably be a little more lively since you'll get the chance to enter the rest of the dialog and learn about conditionals and scripting..

TES Construction Set: NPC Tutorial, Part 3

I'm assuming that you've been following the first sections of the NPC tutorial since you'll need all these skills and more in this part.

Before I launch into full explanations of the nuts and bolts of how to do things, I'll summarize the scenario so that you understand why I'm setting things up the way I am.

When the player is first greeted by Marius, he asks the player to go on a quest to get some cure potions. If the player accepts Marius' quest, Marius gives him 200 gold pieces and a key to Cornelius' cave (and the computer makes a journal entry). If the player refuses the quest, he isn't given anything and a journal entry is made. Once the player has accepted the quest, Marius will immediately ask the player if he's brought the potions on greeting him. Marius asks the player if he'll reconsider and do the quest if the player has previously refused the quest. Marius gives the player a silver longsword once the quest is completed successfully. If the player says he's completed the quest but really hasn't, Marius tells the player to stop taunting him. Many of the conversations with Marius have consequences on Marius' disposition and health. If you bother Marius enough times without completing the quest he'll come to dislike you and possibly even keel over dead.

For Cornelius' part, he greets the player differently than he normally would once the potion quest has been accepted. If the player asks about "sickness" Cornelius suddenly remembers he didn't deliver the potions that he was supposed to and asks the player to deliver them. If the player accepts, Cornelius takes 200 gold pieces and gives the player 3 potions. If the player doesn't accept, Cornelius vaguely mentions that he might get around to delivering the potions sometime.



Preliminary: making a place to store information

In order to make sure that the NPCs give the correct responses when they are talked to, we need a place to store the information about the player's quest status.

Although I could have relied entirely on the player's journal entries to track the quest status, it's also helpful to see how to use a local variable to store values. I decided to use both methods for tracking things so you could see how both work. In many situations, journal entries may be the main method for tracking quest status, though.

Open up Marius' properties sheet and click the button with the 3 dots next to Script. Type in the text shown on the left (you may have to choose File | New from the menu first). The line beginning with the semicolon may wrap around in your web browser but it should be on one line when you type it into the script editor.

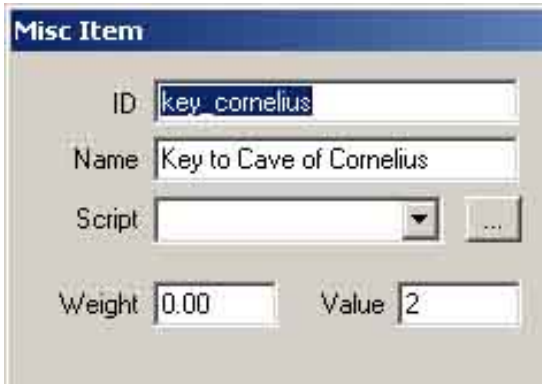
What it does: The Begin MariusObertam and End MariusObertam lines indicate the beginning and end of the script. The short Qstateline indicates we're defining a variable (a storage place) that can store a number between -32,768 and 32,767. The other line beginning with the semicolon is a comment. A comment is just something that doesn't have any function except to indicate something to the programmer looking at the script—the computer doesn't take any action when it sees a comment line.

```
Begin Ma ri usObertam
```

```
short QState
```

```
;Qstate is 0 for Marius not talked to yet, 1  
for quest accepted, 2 for quest refus ed and  
3 for qu est finished
```

```
End MariusObertam
```



When you're done typing the script, save it and close it. All this script does is define a variable that we'll use to track what lines the NPCs deliver. If you look at the comment line, it explains what each of the values represent to us.

Preliminary: creating a key to open Cornelius' door

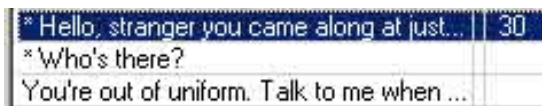
Under the Misc Item tab in the Object Window add a key with the properties shown on the left.

It's probably easiest to open an existing key object, change the information and then click Save. When you save the object you'll be asked if you would like to modify the existing object or create a new object. Be sure you create a new object, not modify an existing object.



After you create this key, set it as the key for the outer door for Cornelius' cave. The key is set up for a specific door from the properties window for the door by selecting the key name from the key drop-down list.

Enter Marius' greeting



The first thing to do is enter the initial greeting in which Marius asks the player to go on a quest for him. Enter this piece of dialog according to the Item #3 information from the dialog sheet that you downloaded in the last section. Be sure to set the conditions and results as specified in the sheet since these items are important. It's also important to add this item towards the bottom of the item list but before any general greetings (we're on the greetings tab).

In the conditions, I'm checking to make sure that this greeting item only applies to Marius, that Marius' disposition must be at least 30 and that Qstate (the local variable we defined) equals 0—which is its default starting value. The fact that Qstate = 0 indicates that this quest has not yet been given to the player.

In the results section you'll notice the command choice "Yes" 1 "No" 2. The choice command gives the player a dialog that makes them choose one of the items before continuing. After the player chooses one of the items, the current topic items are scanned again and the number value of the choice function is set to the corresponding choice so that the dialog can interact with the items the player chooses.

In this example, the player is given the choice of saying either "Yes" or "No" to the quest. If "Yes" is selected then choice is set

to 1 and if "No" is selected then choice is set to 2. Then the current topic category (greeting 1) is scanned again.

* Thank you. Thank you, %PCName. ...	
* Well, if you can't do this for me, I gue...	
* Hello, stranger you came along at just...	30

Sex Disp

Function/Variable

Function	Choice	=	1
Local	QState	=	0

Enter the responses to the choice

Now enter item #4 and item #5 from the dialog sheet as they are listed but enter these items above the greeting you just entered. Notice that I'm using the function Choice and checking its value as a condition for these pieces of dialog. The choice function tells me which number choice was made if a choice was made on the last piece of dialog presented.

Notice that when I'm checking these items I'm looking for the value of Qstate = 0 and choice = 1 or 2. If Qstate = 0 then I know that the quest hasn't been accepted yet, and if choice = 1 or choice = 2 then I know that the player was given a chance to accept or reject the initial quest.

I'm using a few new commands in the results section for each of these:

set Qstate to 2—sets the value of the local variable qstate to 2

moddisposition -5—modifies the Marius' disposition to five less than its current value.

Modcurrenthealth, -10—modifies Marius' current health to reduce it by 10

Journal, XZ_Marius, 10— adds the journal entry in category XZ_Marius with index number 10 to the player's journal.

Player->Additem, "Gold_100" 200—adds 200 pieces of the item Gold_100 to the player's inventory (if the player-> part weren't at the first of the line, it would add the gold to Marius' inventory instead).

Though there are more items in the Results sections than just these items, the rest of the commands in the results section use these same commands with different values so you should be able to figure out what the results items for these pieces of dialog do.

Note: the commands in the results section are executed after the piece of dialog that goes with it has been delivered. You may not enter commands in the result section that span more than one line for each command (if . . . endif statements).

* Please, will you reconsider and run an...	30
* You say you brought the potions, but ...	
* I'm feeling worse and worse all the tim...	
* I am forever in your debt. Take this s...	
* Did you bring the potions I needed?	
* Thank you. Thank you, %PCName.	
* Well, if you can't do this for me, I gue...	
* Hello, stranger you came along at just...	30

Add items 6 through 10 to the greetings section

Now that you've entered a few greetings I'm not going to give you explicit instructions for each greeting since you'll probably be able to figure out how things work from the dialog sheet. Enter each item above the last one in the Info / Responselist. None of the commands are new so you should be able to figure out how these items work.

Note that I change the value of `qstate` in the results sections for many items. By changing the value of `Qstate` I make sure that the correct responses are given to the player depending on the quest status. In item 10, I change the value of `Qstate` back to 0 so that the the original quest responses (to accepting the quest or not) are used. If you don't understand how things work, it's a good idea to pretend like you're the computer and run through the dialog, conditions and results section to figure out exactly what each item does.

"" [You have raised your Urn or Silence]	
* Hello there. Why do you disturb my m...	30
* Who's there?	

Add greeting for Cornelius

You're probably getting used to entering information in the Dialogue Window now. This item is for Cornelius, though, so change the filter for item to filter for Cornelius instead. Look at item #11 from the dialog sheet.

Function/Variable			
Journal	XZ_Marius	>	10

When you enter this item, you'll notice a new condition (as shown on the left). This condition checks the player's journal to see if an entry higher than index 10 exists. If you remember, we assigned index 10 for the entry that says the player didn't accept the quest and index 20 to the index that says the player accepted the quest. The Journal function returns the highest index that exists in the journal under the specified category.

In this case, if the journal category `XZ_Marius` has a value higher than 10 (ie. 20) then the quest has been accepted. If the journal returns a value of 10 or lower for this item then it means that the quest has not been accepted and the player will not get the special greeting from Cornelius with the word "sickness" in it. It's important to introduce the word "sickness" since it will be a topic keyword for the rest of Cornelius' dialog related to the

Topic	V...
-------	------

Adding the rest of the dialog for Cornelius

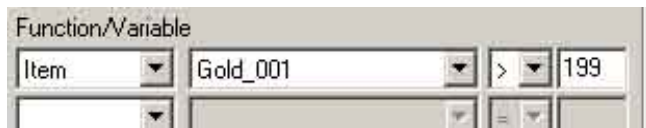
In this section you'll be adding items 12 through 14 from the dialog sheet.

Add a new topic category under the Topic tab. Call the category sickness and add the items that are listed.

scrib	
sickness *	
.....	

You'll notice in this section that I'm using a new condition—the Itemcondition. Cornelius will not offer to allow the player to buy the potions unless the player has at least 200 gold pieces.

Info	Disp
* Ok, I'll run them up myself when I'm n...	
* Here you go then, 3 cure potions. I h...	
* Oh yes, I forgot to deliver the cure pot...	30



Also notice that all these items are in the topic section which means that Cornelius doesn't automatically ask the player if he wants potions unless the player asks about "sickness" first. Notice how this is different from items you put in the greetings section. Any greetings are immediately brought up to the player while items in the topics section must be asked about by the player.

As with the other sections, put each item above the last item you entered in the list. Remember, put specific items above more general items in the list. Also put any items that have choice as one of their conditions before items in the list that use the choice command in their results section.

A Little Trick

With all the things you've done, you might think you were finished with this little quest—but you'd be wrong since I have one more trick up my sleeve. This is a trick that makes things considerably more exciting.

If you were playing this quest, why not just take the 200 gold pieces from Marius, pester him for a while until he keels over dead and then take his loot. After all, he only gives you a silver longsword—which really isn't worth all that much. Moral scruples aside, why not be sneaky and make a better profit?

Well, I'm hoping some people try this quest the profiteering way since I have something slightly nasty and funny in store—a cursed sword (Marius doesn't give you the cursed sword it is just lying around his house). And this is a cursed sword that isn't obviously labeled as cursed so you don't get the typical no-brainer of thinking "Oh, it's cursed, I guess I just won't pick it up." I mean, how likely is it that a cursed item would be THAT obvious. I want the player to have to figure out what's going on instead of just looking at their inventory and dropping the Cursed Sword of Flatulence or whatever the typical geek description would be.

The way I've set up this sword (well actually there are two of them in the world) is that I'm labeling it as Strange Silver Longsword and I'm hoping the player will have a slight clue from the word "strange." The sword is slightly better on reach and speed than a regular silver longsword and it's worth more money, but it also has the property of taking some random actions on the player every 10 seconds or so. It can take one of the following actions on the player:

- It can reduce the player's current health by a random amount.

- It can reduce the player's current magicka by a random amount.

- It can reduce the player's current fatigue by a random amount.

- It can disable the player's ability to fight.

- It can re-enable the player's ability to fight.

- It can disable the player's ability to jump.

- It can re-enable the player's ability to jump.

In testing the Strange Silver Longsword I was laughing with glee at the thought of the poor slob who picks it up and carries it around for a while. It disabled my ability to fight at some key times while rats and other low-level creatures

gnawed on me (if you wait a while the ability comes back because it will randomly re-enable it). It made me randomly fall to my knees in the middle of running somewhere as it sometimes reduced my fatigue below the minimum (I could get back up after a few seconds). It disabled my ability to jump at key times—which can be exciting. I also found I had to rest much more often since it kept chipping slowly away at my health, magicka and fatigue. If someone doesn't notice something strange going on after carrying this longsword around for a while, they need to be a little more observant. Ok, maybe this is my sense of mischief coming out—but I found it kind of funny.

So how did I create this cursed longsword? The answer is "scripting."

Scripting for Fun and Profit (well at least for fun)

For those of you that don't know what scripting is, I'll explain. It's a form of "lite" programming—it's essentially a form of programming that is designed to be easier and/or less functional than a full programming language. It's probably helpful to understand some very basic programming concepts in order to script, but it's not necessary since you may be able to pick up the commands you need without knowing much about programming. I'll explain some basics of scripting and of the cursed sword script, but I'm not going to give you a scripting user's manual—I currently don't have the time or the patience to do this. The scripting commands are (somewhat poorly) documented in the help file for the construction set. I may end up doing more full documentation of the commands in the future sometime.

If you've done any programming before, you may be somewhat confused by the scripting language that the TES Construction Set uses since it's a strange combination of syntax and commands. It has parts that seem similar to a language like Visual Basic and parts that seem similar to something like Java or JavaScript. It also has parts that don't seem to be standard in any way at all. For instance, the command to assign a value to a variable is `set <variable> to <value>`. Why couldn't they use the equals sign as an assignment operator like most other programming languages do? If you know other programming languages, watch out! You'll probably make some mistakes when you use the normal-seeming commands you're used to and the script will not run. You'll have errors in your script because the scripting language forces you to do things in a way that feels strange to you. If you're used to a programming language that handles or raises events for you, you'll also be bothered since the scripting language does not do this.

A script is generally attached to an object in the game (such as an NPC, a weapon, a key). When the player is within the same cell or within a certain distance of the object with the script, the script runs—over, and over, and over again. In fact, the script keeps running more or less continuously the whole time until the player moves far enough away from the object with the script. It's important to make your script end as quickly as it can on each cycle so that it doesn't suck more processing power away from the main game and other items than it needs to (it could make things run more sluggishly if you're not careful with your script coding). In other words, if you can determine that the script doesn't need to run on that cycle in the first few lines of code, it is good to exit the script immediately once you've figured this out.

Another thing is that the documentation for the scripting language suggests that commas must go in many places when issuing a command or using a function. Some of the places they put commas (such as right after the command name) are very strange. In my experience, you can use either a space or a comma to separate items since the scripting language will parse either as a separator in most cases. The scripting language seems to be case-insensitive (so that TheBob would be the same thing as thebob or THEBOB). It also chokes if you have more than one function per line of code (it can have more variables and other items per line, though).

Well, enough background. Here comes the script for the cursed longsword.

Create the new object

To create the object, find the normal silver longsword in the weapons category in the object window. Open it up and change the id to "cursed silver longsword." Change the name to "Strange Silver Longsword."

Enter the other values as shown in the screen shots or customize them to

Weapon

ID:

Name:

Type:

Script: ...

Weight: Value:

Health: Speed:

Enchantment: Reach:

Enchanting:

☒ Ignores Normal Weapon Resistance

your taste. You will not be able to change the script drop-down list yet, though, because we haven't put the script for the silver longsword in yet. That's what's coming up in the next steps.

Damage

	Minimum	Maximum
Chop	<input type="text" value="2"/>	<input type="text" value="14"/>
Slash	<input type="text" value="1"/>	<input type="text" value="20"/>
Thrust	<input type="text" value="4"/>	<input type="text" value="18"/>

☐ References Persist

☐ Blocked

Create a new script



Click on the ... button next to the Script drop-down list. When the script editor opens up, click on Script | New to open up a blank script window.

Opening the script and defining some variables

Begin CurseSilverLSword

short rndresult

short rndamt

float timer

To start out the script, use the Begin <scriptname> statement. This indicates the beginning of a script and the script name.

The next thing I do is define three variables. The first two are short variables—which means that they can store a number between -32,768 to 32,767. A float variable is a floating point variable—that is it is a variable that contains a decimal point and can hold very large or small numbers. A long variable can hold values approximately between negative 2 billion and 2 billion.

float or a long to store a number between 1 and 100 would take up unnecessary storage space and could also slow down the calculations being made.

I'm going to store a random number in the `rndresult` variable (to choose an action randomly) and store a random number in the `rndamt` variable that will choose how strong an effect is.

```
;see if player is carrying the sword  
  
If ( play er->GetItemCount, "cursed  
silver longsword"==0 )  
  
Return  
  
Endif
```

Telling the script to exit unless it is needed

There are some comments and some `if . . . endif` statements here. The `if . . . endif` statements evaluate a condition and execute the statements that come before the `endif` statement when the condition is true. Notice I use double equals signs (`==`) when comparing to see if something is equal. Although the 2nd and 3rd lines in this section appear as two lines, they should appear as one line in the script editor (they appear this way because they wrap around).

```
;return if in menu mode  
  
if (MenuMode==1)  
  
return  
  
Endif
```

The `return` statement tells the script to exit (for this pass) without doing any further processing.

In this portion, I'm telling the script to exit immediately if the player does not have the "cursed silver longsword" in his inventory. I'm also telling the script to exit if the player has a menu up on the screen (this prevents the curse from happening while the game is "paused" because a menu is up).

```
set timer to ( timer +  
GetSecondsPassed )  
  
if (timer<10)  
  
return  
  
endif
```

Keeping track of time

The scripting language doesn't provide an easy way to keep track of elapsed time for an event. About the only command the scripting language has that keeps track of time is the `GetSecondsPassed` function. It returns the time that has passed between now and the last time the script ran. The documentation refers to this time difference as the time between frames.

In order to get the time that has passed for a longer period than a frame, I need to keep adding time to the timer variable each time the script runs. The first line performs the function of adding the latest frame time to the cumulative frame time that I'm storing in the variable named `timer`.

The `if . . . endif` statement tells the script to exit if the time added up is less than 10 seconds. I only want to take an action every 10 seconds if the player is carrying the cursed sword.

```
set timer to 0
```

Resetting the timer and choosing some random numbers

```
set rndresult to random,11  
  
set r ndamt to random, 20
```

The first line resets the timer variable to 0 so the counting can start again. Remember these lines only happen if the script has not exited from a `return` statement earlier in the script.

```
set rndamt to ( 0 - rndamt - 10 )
```

I'm having the computer put a random number between 0 and 10 in the variable rndresult (there are 11 possibilities if you count from 0 to 10).

```
;messagebox, "Strange = %.0f",  
rndresult
```

The computer is then putting a random number between 0 and 19 in the rndamt variable. I'm then changing this positive number to a negative one between -10 and -30 with the next line. If you're wondering why I did things this strange way, it's because the scripting engine was having trouble when I phrased the syntax in a more normal way (such as set rndamt to -rndamt).

The next line is commented out (it doesn't execute because of the semicolon in front of the line). If you take the semicolon off of the line, it will pop up a message box telling the player what random result number has been selected every 10 seconds. I put this line in for troubleshooting. It's also fun to run it with this line uncommented to see what is happening to you when you're the player carrying the sword around.

```
if (rndresult==0)
```

The random results that may happen

```
player-> modcurrenthealth, rndamt  
endif
```

These are the results that may happen based on the value chosen for rndresult

0 = the player's health is decreased

```
if (rndresult==1)
```

1 = the player's magicka is decreased

```
player-> modcurrentmagicka, rndamt  
Endif
```

2 = the player's fatigue is decreased in value (this decreases 3x more than the other two would)

3 = the player's fighting controls are disabled

```
if (rndresult==2)
```

4-6 = the player's fighting controls are enabled

```
set rndamt to ( rndamt * 3 )
```

7 = the player's jumping control is disabled

```
player-> modcurrentfatigue, rndamt  
endif
```

8-10 = player's jumping control is enabled

```
if (rndresult==3)
```

```
DisablePlayerFighting
```

```
Endif
```

Notice that some of the sections have one if...endif statement inside another one. This syntax is a way of ensuring that both conditions are met. As far as I could tell, the scripting language doesn't have a normal "and" logical operator--you can get the same result this way. Also values 4-6 and 8-10 do not change anything unless either the jump control or fighting has been previously disabled—because of this, the script doesn't have a negative effect except for about 1/2 of the time (5 out of 11 times to be exact).

```
if (rndresult>=4)
```

The last line tells the script that it is finished. This type of ending line finishes the script.

```
if (rndresult<=6)
```

After the script is typed in, save it and make sure it shows up as the script that cursed silver longsword is using.

```
EnablePlayerFighting
```

Endif

Endif

if (rndresult==7)

DisablePlayer Jumping

Endif

if (rnd result>7)

EnablePlayerJumping

Endif

End CurseSilverLSword

Put the longsword in the world

The last thing I did was to put the longsword two places in the world—I put one in Marius' inventory (so if the player kills Marius he can pick it up off his corpse). I also put one on the bed in the lighthouse so that the player can just sneak and steal it if he wants.

I also set the owner on both items so they are owned by Marius.

Now that the plugin is finished, you may [download it](#) to test things out. Remember you can get to the island in the game by typing coe -10, -5 from the command prompt in the game (press ~ first).