

Introduction and Guide to Scripting in Neverwinter Nights

Contributors marked as known with their work
Compiled by Travis Addington (Papermonk)

Ver 1.0
11/24/2005

Questions, comments? t.addington@gmail.com

Introduction

There's no doubt than I'm an odd person to be putting this together. And no doubt something like this is coming a little "late in the game," so to speak. Yet, that is exactly why I compiled this information. I'm learning (finally) to script without the help of tools and finding concise and consolidated information was near impossible. It was scattered all over the bioboards and while that is a great resource in and of itself the information is scattered and can be a bit hard to pin down.

What I've tried to do with this document is to collect all the important information a beginner or expert scripter may need. I've plundered tutorials and sticky posts for relevant information, inserted some standards into appearance, spell checks, and organized it.

It is important to note that **none of this work** is mine. This is like the CEP of Scripting information, I just compiled it.

Now, no doubt I missed important information. If so, point it out.

Also, some of this information is old and not being proficient myself I have no way of knowing what may be out of date or trumped by later patches. If you find something like this, let me know and I'll fix it.

As for the late in the game... well, it's never to late to learn. A great deal of the basics of NWScript will no doubt still be useful for NWN2 and I will attempt to keep this up to date when it comes out.

I'd like to thank all the people who took the time to write out this information, the people who helped me go through this, and just the NWN Community and Bioware for the help and support over the years in all my various endeavors.

I. The Basics - Scripting 101

I. The Basics - Scripting 101.....	3
Celowin's Scripting Tutorial Lesson I – The Basics	6
Celowin's Scripting Tutorial Lesson II – Local Variables	11
SETTING VARIABLES.....	16
DEFINING AN OBJECT	21
CREATING OBJECTS	25
REGARDING ANIMATIONS	27
EXPLAINING WAYPOINTS AND WALKWAYPOINTS	32
MORE FULLY-COMMENTED ONSPAWN SCRIPT	39
REWARDING GOLD AND XP	41
MAKING SKILL CHECKS	43
SITTING IN CHAIRS AND SLEEPING	48
HOW DO I MAKE MY NPC'S INITIATE DIALOGUE ON THEIR OWN?	51
HOW DO I MAKE MY NPC ATTACK THE PC HE IS TALKING TO?	55
HOW DO I START MY STORE?	57
USING MODULE EVENTS	59
USING CREATURE EVENTS.....	63
USING TRIGGER EVENTS	70
QUEST CREATION WALKTHROUGH - NO SCRIPTING REQUIRED.....	72
Celowin's Scripting Tutorial Lesson III - Conditionals	76
ADDING & REMOVING EFFECTS.....	84
Celowin's Scripting Tutorial Lesson IV - UserDefined Events	89
USING DIALOGUE WITH PLACEABLES AND TRIGGERS	98
ERFs : WHAT THEY ARE AND WHAT THEY'RE GOOD FOR.....	105
FACTIONS, SHOUTS AND ATTACKING MY ENEMY	108
Celowin's Scripting Tutorial Lesson V - NonNPC Scripts	118
Celowin's Scripting Tutorial Lesson VI - Loops.....	127
Celowin's Scripting Tutorial Lesson VII – Scripting for Items.....	135
Celowin's Scripting Tutorial Lesson VIII - Functions.....	141
Celowin's Scripting Tutorial Lesson IX - More on Functions: Functions that Return, Default Parameters, Libraries.....	149
Celowin's Scripting Tutorial Lesson X - Henchmen	160
Celowin's Scripting Tutorial Lesson XI - Process Example: Bar Brawl	172
Huntsmans Guide To Henchmen (Part 1) - SOU / HOTU Style	194
Huntsmans Guide To Henchmen (Part 2) - One Liners, Interjections and module transfers.	198
Introduction to struct	206
Get2daString and Loops.....	211
The SoU Treasure System	214
III. HOU Misc Updates.....	217
Post-SOU Doors – Old ones don't work	218
Script Caching.....	219
Fun With Petrify.....	219

Adding Crafting Materials to Your Campaign.....	220
How to add more dyekits	220
Prestige Class Control Variables	222
New scripting functions spotlight.....	223
An introduction to Tile Magic	225
Apply/Remove Tilemagic from Specific Tiles	226
On Hit Cast Spell	229
Intelligent Weapons.....	232
Adding Material Components to the Crafting Feats	233
How to Disable Crafting Feats.....	234
HOU Problems With Scripts That Use the DelayCommand.....	235
Dynamic item properties.....	235
HotU OnSpawn and Variables	244
http://www.statman.info/conversions/hexadecimal.html	251
Antimagic Tutorial	251
HotU Wandering Monster System	251
Ambient Simulation System	254
VI. Appendix I.....	255
How do I close a door automatically after it has been opened?	255
How do I make my doors Close and lock?	255
How do I make my NPC Open / Close a Door?	256
How do I make a portal?	256
How do I make my NPC sit.....	257
How do I make my NPCs stay seated while talking	257
How do I take gold from a PC in a conversation? How do I make sure he has enough to Pay?.....	258
How do I make my NPC recognize me when I talk to him a second time.	259
Why is CreateObject giving me badgers?.....	259
How do I let PCs sit in chairs?.....	260
How do I make a Combat Dummy?.....	260
How can PCs gain XP for training at a Combat Dummy / Target.	261
How do I get an NPC to take more than one of the same item from a PC's inventory?.....	263
How do I get a NPC to leave after I finish a conversation?	264
How do I get an NPC to Check for more than 1 of the same item in a PC's inventory? option 2	264
How do I make sure the PCs have a certain Item in possession before they are allowed to rest?.....	265
How do I show the Entire Map to the PC if they buy a map from a merchant like in the OC?.....	266
How can I make it so that after the conversation the NPC will turn back and face a certain Direction.....	267
<i>How do you use Waypoints with NPCs?</i>	268
What function do "#include" or "<span..." do??	269
How do you give each entering PC an Item as they enter the Module for the First time?	271

Can Waypoints Run Scripts?	271
I have a script that Adds an Effect to a PC. How do I remove it?	271
What does "!" mean in a Script?	272
What's the difference between sDest and oidDest?	272
How can I make my NPC sit Cross-legged and NOT get up when talked to?..	272
How does the OnHeartbeat Function?	274
I have a script in the "Actions Taken" at the end of a Conversation, but when the conversation end the NPC just stands there as if the script is not triggered.	274
What is a switch/case ?	275
I just re-installed NWN. Do I have to re-play all the chapters again to unlock them? AND How can I open all the chapters in my Toolset without having to play them all?	278
What's the Basics for making a Merchant?	278
Who do I make my NPC say something while in combat?	279
How could I easily add multiple journal entries upon the player entering the module?	279
How do I know what Journal Entry the PC has?	280
What's the fastest way to tie the respawn of a PC to a specific way-point?	280
How do I make a lever actually do something?	281
How do I strip a PC of all Items and Gold?	281
How do I make a placable start a conversation with a PC on Use?	282
How do I make a NPC "flee" the area after a conversation?	282
How do I make Floaty Text, or Speech on objects and doors?	283
How can I make a "Level up" Script?	283
How do I use the Special Conversations mentioned in the OnSpawn Script?	284
How do I make my NPC turn and face there previous facing after being clicked on?	285
How do I make my NPCs motionless?	285
How can I restrict how often PC's can rest?	286
How do I combine several scripts to activate on the same event, such as the OnHeartbeat?	287
Is there a way to define which immobile animations you want an NPC to do?	288
How do I make a sign use a floaty for its name?	288
How do I make the corpse face up?	289
Okay, now how to I bring the corpse back to life? (he-he-he)	290
Life after death or Bioware dying, death, and Respawn system.....	290
What does the ? mean in NW Script?	299
VII. Appendix II: Code Error Explanation	301
VIII. Appendix III: Waypoints Explained	309
VIII. Appendix III: Important Resources	314

Celowin's Scripting Tutorial Lesson I – The Basics

Introduction

The purpose of this sequence of lessons is to take a complete beginner to programming, and teach him or her how to use NWScript to write modules. The early lessons will be very basic, and anyone that has done any coding at all will be able to skip over them. The goal here is to make the lessons so that even the people that just shudder at any type of code can learn.

Feel free to post these lessons on any forum, print them out, or modify them. However, just give me credit for doing them.

Any comments on these lessons, good or bad, can be sent to me at james.foxglove@verizon.net

I am going to assume that anyone looking at these lessons has at least played around with the Aurora Toolset a bit. If there is enough feedback that people don't know how to do the simple placements that I have in these lessons, I will consider spelling out in more detail what needs to be done.

Let's Begin

Open the toolset, and create a new module using the wizard... we'll call it "Test Module". Create a new area, call it "Test Area 001". Use whatever size and tileset appeals to you, for the purposes of this lesson it really doesn't matter. (You probably want to keep it small in size, for ease of finding things. I suggest 2x2.)

Use the "paint creatures" tool to lay down an npc. Give it whatever appearance you wish.

Now, do all the following:

- Right click on the npc you just put down.
- Click on "Properties" from the menu that comes up.
- Change the tag of the npc to SINGER
- Go to the "Scripts" tab.
- There are default scripts in every slot. Click on each slot, and delete the script name.
- Click on the "edit" button next to the "On Heartbeat" slot.
- Type in the following script (Every character needs to be exactly the same.):

NWScript:

```
void main()
{
    ClearAllActions();
    ActionSpeakString("This is the song that never ends.");
    ActionWait(1.5);
    ActionSpeakString("Yes it goes on and on, my friends.");
    ActionWait(1.5);
    ActionSpeakString("Some people started singing it not knowing
what it was.");
    ActionWait(1.5);
}
```

```
        ActionSpeakString("And now they'll keep on singing it  
forever, just because...");  
    }
```

- I recommend you actually type it in instead of cutting and pasting. You will learn more about the way the script is written. If you make a mistake, so much the better.... you will see how important every character can be.

- Click on "Save As" and call it tm_singer_hb

- Check the window at the bottom of the screen. It should say something like: "0 Errors. 'tm_singer_hb' Compiled successfully." If it doesn't, you made a mistake typing. Double-check yourself.

- Close the script edit window.

- Click "OK" on your npc window.

- Save your module.

- Call up the game, and go to "Other Modules" to start up what you've done.

- Your npc should be singing the little ditty over and over and over...

Analyzing It

Now for the hard part... I'm going to go step by step through what we did, and try to explain everything. I'll do this in a "question and answer" format, trying to guess at everything someone might ask.

Why did you call it "Test Module"?

Really, in this case, there is no reason for it, aside from making it something easily recognized when you go to the Other Modules screen.

Why did you call it "Test Area 001"? Wasn't the default name the toolset assigned good enough?

For our little test module, it probably doesn't matter. However, it is a good habit to get into to rename your areas, for purposes of exporting. If you want to take just one area out of your module, and put it into a different one, it has to have a unique name. If everyone uses "Area 001" in their module, then importing and exporting areas is problematic.

Why did you give the npc the tag SINGER? Why all capitals?

For a few things, the tag needs to be all capitals. You may never run into this, but it is again, a good habit to get into.

Also, I recommend giving every creature you paint a simple, easy to remember tag. SINGER describes the npc perfectly, so it is easy to refer to him in a more complicated script.

You also want to keep your tags short. I recommend 8 characters maximum.

Why did we delete out all the default scripts? Why do they show up there if we are just going to delete them?

The default scripts define a number of behaviors that we don't want to deal with quite yet. For example, depending on what npc you originally painted, it may well have ended up attacking you when you entered the module. We deleted the other scripts just to keep things simple – we want our npc to sing the song, and nothing else.

In later lessons, I'll go into making better use of the default scripts.

Why did we put our script in the "On Heartbeat" slot? Why are there so many slots anyway?

Each one of these slots for scripts defines a time when that particular script is called. The "On Heartbeat" slot calls the script every six seconds. This is why the npc sings the song over and over... every six seconds, he sings his four lines.

There are uses for every one of the script slots, depending on the behavior you want. In fact, the "On Heartbeat" really should be your least used script slot. Too many scripts firing every six seconds can cause performance issues with your computer.

What is this "void main()?"

This looks rather simple, and the fact that it shows up exactly like this in nearly every script causes many people to just blow it off. I'm going to attempt to explain it fully here, but I can't guarantee it will make complete sense until later lessons.

NWScript is coded using "functions," which tell the program what to do. Some functions are pre-written for us, and we just put them into our code. However, when we are making scripts, we are in fact writing our own functions. This line is sort of "setting up" our function.

First the "void". This tells the script what kind of "answer" will come out of the function. Our little singer is performing actions, but isn't calculating out any sort of an answer. So the "void" says that the function doesn't actually give an answer.

The "main" says that it is the main part of our script. We can write other functions into our script, but the "main" one is the part that is called when the script is started.

In between the "(" and the ")" is what kind of input is coming into our script. Again, our script doesn't need any input, so there is nothing between them. We always need the (and), even without any inputs.

What about the { and }?

These are used to set off sections of the script. In this case, it is used to say that everything we have written is part of our "main".

Why does every line inside the main end with ";"?

Basically, to tell the script that it is the end of the line. Some complex instructions won't fit well on one line. Breaking it up onto separate lines is fine, the program just treats it as one line until it comes to the semicolon.

You can think of it very much like the period in the English language. (The reason the semicolon is used instead of a period is to prevent confusion with decimal points.)

What does the ClearAllActions() do?

This is a tricky one, because in this case it actually doesn't do anything. It is put in more as insurance, hoping that it never actually has a job to perform.

Again, I'll try to explain... every other line in the script is an "Action" command. The npc doing the actions will do them one at a time, making sure to finish one before it goes on to the next.

But now remember that our function is assigned to the "On Heartbeat" and will be called every six seconds. We have seven actions that we want the npc to do, what if he only finishes 4 of them in the six seconds? There are still 3 actions left to perform, and then the npc is told to do another 7 actions... so he has 10 things to do now. He does another 4, and then a new set of 7 instructions comes in... 13 to do now. He will fall further and further behind. Eventually, this will cause problems.

There are better ways of dealing with this. But for now, it is easiest just to tell the npc to "forget about everything you still had to do." This is what the ClearAllActions() is for.

If you want to experiment, you edit the script, and change the number inside all the ActionWait commands from 1.5 to 3.0. You'll see that the npc can't finish the whole song before he starts it back up again.

While I'm here, let's tie this back with a bit we were discussing a question or two back. The ClearAllActions is a function, just like main. So once again, having nothing between the (and) means there is no input to this function.

What about all these other lines in main?

These are all the instructions telling the npc what to do. I think these are pretty obvious, but I'll try to explain them a bit anyway.

ActionSpeakString has the npc say something. Again, we have a function, but this one actually does take an input... the text that the npc is going to say. This text is called a

"string", and similar to the English language, needs to be encapsulated by quotation marks.

ActionWait has the npc sit and do nothing for a number of seconds equal to the number put into it. So the ActionWait(1.5) has the npc wait for 1.5 seconds between each thing that is spoken.

Why name the script tm_singer_hb?

Again, you could call it just about anything, but you want to call it something easy to remember. Standardization is the key.

We use tm to stand for "Test Module." Any script written for test module we will start with this two-letter code.

"singer" is of course the npc we are attaching the script to. Even though the npc tag was all capitals, we use all lower case letters for script names. (Note: this is why I recommend using tags for objects of 8 characters or less... otherwise these script names can get very long.)

"hb" says that this script is for the heartbeat of the npc.

Why isn't this lesson any longer?

I fully realize that with just this first lesson, you can't yet do a whole lot of scripting on your own. However, I'd rather do too little than too much, and have people get frustrated with the amount of information they need to absorb. Give me a couple of days, and maybe I'll get another lesson cranked out.

[Edited By David Gaider: Friday, 05 July 02:54AM (GMT)]

Celowin's Scripting Tutorial Lesson II – Local Variables

Introduction

The purpose of this sequence of lessons is to take a complete beginner to programming, and teach him or her how to use NWScript to write modules. The early lessons will be very basic, and anyone that has done any coding at all will be able to skip over them. The goal here is to make the lessons so that even the people that just shudder at any type of code can learn.

Feel free to post these lessons on any forum, print them out, or modify them. However, just give me credit for doing them.

Any comments on these lessons, good or bad, can be sent to me at james.foxglove@verizon.net

I am going to assume that anyone looking at these lessons has at least played around with the Aurora Toolset a bit. If there is enough feedback that people don't know how to do the simple placements that I have in these lessons, I will consider spelling out in more detail what needs to be done.

For those of you that missed earlier lessons and are interested in looking them up: [Lesson 1](#)

Let's Begin

This lesson will likely be a bit longer than the first one, and deal with some more advanced concepts. I'm not certain how well I'll be explaining things here, so *please* ask questions if you don't understand something. Odds are if what I write isn't clear to you, there are five other people that want to know the same thing.

Let's just modify the module we started before. Open it up in the editor.

Right click on the npc we have, go to properties, and go to the script tab. (We could make a new npc, but do we really want this one chanting his song in the background? It gets old really quickly....)

We're going to need two scripts this time, both short.

The first one goes into the "OnSpawn" handle.

NWScript:

```
void main()
{
    SetLocalInt(OBJECT_SELF, "SINGER_COUNT", 0);
}
```

Save it as `tm_singer_os` (os for OnSpawn)

Now, even though this is only one line, it is things like this that tend to freak out people that have never scripted before. So, I'm going to explain this one before I go on to the "real" script.

First off, note that we're putting this on into a different handle than before. Any script attached to "OnSpawn" is run exactly once... when the npc first loads into the game. Because of this, it is ideal for initializing things, which is what we are doing here.

This one line sets up a *variable*. Think back to your last math course, what was a variable? A letter which represents a number. That is pretty much the same thing that we are doing here, except that instead of using a single letter, we give it a longer name.

The command `SetLocalInt` is the function that is setting everything up. It takes in three inputs, separated by commas.

- The first is the thing which the variable is "attached" to.
- The second is the name we are giving to the variable.
- The third and final is the value we are putting into the variable.

At this point, I hope the second and third of these make sense... we are calling our variable `SINGER_COUNT` and we are giving it the value of 0. It is the first input that takes a bit more explanation.

Nearly everything in a module is an object. NPCs are objects. Placeables are objects. Waypoints are objects. Even the pcs themselves are objects. With so many objects floating around in a module, the tricky part becomes making sure you are referring to the right one.

`OBJECT_SELF` is one of the most handy ways to refer to an object. As you might guess, it is used to refer to whatever object is calling it... in this case, our script was attached to an npc, so that npc is what is meant by `OBJECT_SELF`.

Putting all this together... this one line has defined a variable called `SINGER_COUNT`, given the variable a value of 0, and stored it with the npc `SINGER`.

Also of note is that the `Int` in `SetLocalInt` stands for "Integer", or basically a whole number. We can set it to 4, or 0, or -35, but we can't set it to 3.8, for example.

Now, let's go on to the second script. We'll overwrite the `OnHeartbeat` script from the last time.

NWScript:

```
int nCount=GetLocalInt(OBJECT_SELF, "SINGER_COUNT");  
void main()  
{
```

```
        nCount = nCount+1;
        ActionSpeakString("I have spoken "+IntToString(nCount)+"
times.");
        SetLocalInt(OBJECT_SELF, "SINGER_COUNT", nCount);
    }
```

Save it, keeping the name `tm_singer_hb`

Before I explain this, it is probably a good idea to take a look at what it actually does. So close your script window, click OK on your npc, and save the module. Start up the test module, and see how the npc behaves.

This is the first script I have done where we have had anything before the `void main()`, so it is worth mentioning. Everything before that is called the "initialization" of the script. Basically, it sets up things that the script can refer to. Unlike the local variable that we stored before, this `nCount` only stays around as long as the script is running. As soon as the script finishes, it gets rid of `nCount`. For this reason, the variables created in a script like this are called "temporary variables."

Notice how many times we used "`nCount`" inside the body of `main`. Imagine how long and confusing the script would be if every time we had to write something like `GetLocalInt(OBJECT_SELF, "SINGER_COUNT")` instead of just `nCount`. Pretty sick, eh?

Now, the name `nCount` is another bit of standardization. Any time you declare a temporary variable, the first letter is a tag to point out what kind of value it takes. The `n` stands for integer. According to the code, you could give it any name you want, but by naming it this way, you know immediately that it must take an integer value. The "Count" part of the name tells you a bit about what it will be used for.

The "`int`" says we are defining an integer variable. The "`=`" means "give it the value of." And the `GetLocalInt` retrieves the value of the variable `SINGER_COUNT`.

Note the similarity of the functions... we used `SetLocalInt` to set up and store the local variable, and `GetLocalInt` to retrieve the value.

So, that very first line does a lot. It creates a temporary variable `nCount` that the script can use, and right away gives it the value of the variable assigned to `SINGER_COUNT` and stored on our npc. So, the first time this is called, `nCount` has a value of 0, since that is what we initialized it to in our `OnSpawn` script.

I know, this is a lot of explanation for one line of code, and I doubt anyone would digest it all on one pass. Read through it again, and if it still isn't clear, don't worry too much about it. This kind of thing makes a lot more sense once you've done a few examples.

Let's move on to the main body of the script, then. Consider the first line:

```
nCount = nCount+1;
```

This just takes our temporary variable, and increases it by one. For expressions like this, it helps some people to read the word "set" in front of the expression. So in words, this might be read "set the variable nCount equal to the value of nCount plus one."

If nCount starts out with a value of 9, this will set it to have a new value of 10.

(As an aside, another way this can be put into NWScript is to write the line as

```
nCount++;
```

The ++ stands for "increment", that is, increase by one. This is fine once you are used to it, but is generally a bit more confusing for people that are just learning.)

The second line is an ActionSpeakString, but the inside is kind of funny. I've done a few things here... first, notice that we have three things on the inside, separated with + signs. When applied to strings (text), the + just adds one string on to the other. So for a simple example:

```
"This is a string." would be exactly the same as "This is "+"a string."
```

There would be no reason to do something like in the above line. But it comes in handy when we introduce the middle part of the expression: IntToString(nCount). This does exactly what it says... nCount is an integer. This changes the numerical value that it has into a string, or bit of text.

This idea can be a bit confusing for a non-coder, so let me try to make an analogy. Suppose I say something silly like "an elephant wearing a tutu." You immediately get a picture in your head, and don't think about the individual letters that go into making up the phrase I said. It is only when you go to write it out that you start thinking a-n-space-e-l-e-p- and so on.

The computer is the same way. It is storing the value of the variable, and it isn't thinking about how to "write it out." The IntToString tells it to do that conversion to text so that the npc can speak it.

The final line you should understand... it just stores the new value of nCount back into our local variable. Remember that the actual nCount is only temporary, and will be discarded once the script ends. So if we want the variable to update, we need to store it again.

Preview of Lesson Three

Lesson two was originally going to be two or three times as long, but I think I'm going to split it up into separate lessons. However, I just can't leave our last script in the state it is

in... it has an error that just irks me.

If you are detail oriented, you might have cringed a bit when you first tested the script. The very first words out of the npc were "I have spoken 1 times." Easy enough to understand what is meant, but not the height of grammar, either.

The way to get around this is by using a "conditional" or "if-statement". I'm not going to explain this all until Lesson Three, but here is a script that fixes the problem. Just replace the heartbeat script with this.

NWScript:

```
int nCount=GetLocalInt(OBJECT_SELF, "SINGER_COUNT");
void main()
{
    nCount = nCount+1;
    if (nCount==1)
    {
        ActionSpeakString("This is the first time I have spoken.");
    }
    else
    {
        ActionSpeakString("I have spoken "+IntToString(nCount)+"
times.");
    }
    SetLocalInt(OBJECT_SELF, "SINGER_COUNT", nCount);
}
```

Hopefully, even without explanation, this is somewhat easy to understand. And really, once you fully understand it, you are probably 80% of the way to writing your own scripts.

[Edited By David Gaider: Friday, 05 July 02:53AM (GMT)]

SETTING VARIABLES

One of the most common things you will do is set variables and retrieve them... this is the only way of determining whether most events have taken place or of changing/keeping track of different story states.

The most common command you will use when setting variables is this:

```
void SetLocalInt(object oObject, string sVarName, int nValue)
```

This command sets a 'local integer'. What does it mean by local? It means that the integer is referenced to, or 'stored on', the oObject you specify (it isn't really stored there physically, but it's easier to think of it that way).

There are three things that are important to remember, here:

- the 'oObject' has to be a valid object. You will see many commands that return **object** at the beginning of their structure (like it says 'void' at the beginning of SetLocalInt). This means that those commands return data of the type object.

Whatever object you specify is where the integer is referenced to. If you're in dialogue, using *GetPCSpeaker()* will return the PC who's talking to the NPC as the object. You can use *GetModule()* to store the integer on the module itself... you could use OBJECT_SELF to reference whatever object is running the current script, etc.

This is important because the variable you choose becomes unique to that object. If you store a variable by the name of "VARIABLE1" with the value of 1 on the module, that's the only variable by that name that can be stored there. You could have a "VARIABLE1" stored on a different object, like a PC, and it could have a completely different value.

- the 'string sVarName' is the name you give to your variable. Make sure it is unique. Like all strings, it must be enclosed in quotes (like "VARIABLE1" above). Variable names are not case-sensitive.

- the 'int nValue' is the information you're storing in the variable. Since it is an integer we're setting with this command, it must be a number (with no decimal places)... such as 1, 15 or 320. If you wanted to store different information here, you need a different command:

SetLocalFloat = stores a float number (a number with a decimal place)

SetLocalString = stores a string (such as a sentence or word or an object's name... it must have quotes around it, just like the variable name)

SetLocalLocation = stores a location

SetLocalObject = stores object data (such as a placeable object, an inventory item or a creature)

How do I use variables?

Well, once you've set a variable on something, it stays there until replaced or destroyed. At any other time in another script (or the same script) you can call on that variable again and recall the information stored in it.

If I wanted to recall that integer data I just set in a variable, I would use this:

```
int GetLocalInt(object oObject, string sVarName)
```

This is exactly the same command as what you just used to set the variable... minus the actual value of the data (as that's what you're looking for).

Note that this isn't a void command... it returns an integer, actual information. This means you don't just use this command on its own. You use it to define either needed 'int' data in another command or a variable you are setting... like so:

```
int nValue = GetLocalInt(OBJECT_SELF, "VARIABLE1");
```

The above means you have just specified a new variable called 'nValue' which is now equal to the integer that was stored in the OBJECT_SELF under the name of "VARIABLE1".

Confused yet? Examples...

One great example of how a variable might be used is to track the progression of plots. Let's say I want to have a variable called "STORY"... when the player accepts the plot, I set it to 1. When the player finishes the plot, I set it to 2.

The player accepted the plot...

At the point in dialogue where the player accepts, I go to 'Actions Taken' and make a script that sets the "STORY" variable to 1 on the PC.

NWScript:

```
void main()  
{  
    SetLocalInt(GetPCSpeaker(), "STORY", 1);  
}
```

The player finishes the plot...

Let's say the player has returned, having done what the NPC wanted. The NPC congratulates him and the "STORY" variable is set to 2 on the PC.

NWScript:

```
void main()
```

```
{
SetLocalInt(GetPCSpeaker(), "STORY", 2);
}
```

How do I know if the player has accepted or finished the plot or not?

When it comes to dialogue, you just need to make a new starting node for each condition. The computer will start from the top and check any scripts in the 'Text Appears When' section and see if it is TRUE (or if there is no script there, which makes it automatically TRUE). If so, it starts that node. If not, it continues to the next.

So, with this plot, you'd list the nodes from last to first...

- #1. (plot done) "Thank you for doing that plot."
- #2. (plot accepted) "Have you finished that plot?"
- #3. (not accepted, not done) "Hello there!"

So you need a script which returns TRUE for #1 if the PC has the "STORY" variable on him set to 2, right? And for #2, the script must return TRUE if the variable is set to 1. No script is needed for 3, because if both #1 and #2 were FALSE, then the plot has neither been accepted nor done.

Is the plot done?

NWScript:

```
int StartingConditional()
{
int nDone = GetLocalInt(GetPCSpeaker(), "STORY") == 2;
return nDone;
}
```

Is the plot accepted?

NWScript:

```
int StartingConditional()
{
int nAccept = GetLocalInt(GetPCSpeaker(), "STORY") == 1;
return nAccept;
}
```

So you'd end up with this:

- #1. (1st script in "Text Appears When") "Thank you for doing that plot."
- #2. (2nd script in "Text Appears When") "Have you finished that plot?"
- #3. (no script) "Hello there!"

In the above script, I only want the NPC to give the plot to 1 person.

At the point when he sets the "STORY" variable on the PC to 1, why not have him set a variable called "PLOT_GIVEN" to 1 on OBJECT_SELF (the NPC)?

Then when the dialogue branches from the default #3 dialogue (because another PC with no "STORY" variable on him wanders along and speaks to the NPC), you could have the dialogue split into two:

```
|
|
--> (script in "Text Appears When" that returns true if
"PLOT_GIVEN" on OBJECT_SELF returns a 1) "Sorry, no work for you
today."
|
|
--> (no script) "How would you like to do a plot for me?"
```

A REALLY SIMPLE EXAMPLE QUEST DIALOGUE

The thing to remember is that when you line up your 'starting nodes' underneath the Root, they should be in reverse order of completion.

Let's say your NPC gave three jobs. Imagine the following structure under the root:

- #1. "I don't have any more jobs for you."
- #2. "Are you finished job #3?"
- #3. "Are you finished job #2?"
- #4. "Are you finished job #1?"
- #5. "Would you like a job?"
- #6. "Hello!"

The thing to remember is that when the NPC is clicked on for dialogue, the computer will begin scanning the scripts in the 'Text Appears When' part of these nodes. If the script returns TRUE (or there is no script), that node is initiated. If it returns FALSE, it continues onto the next node.

The following is a rundown of what scripts would be at each node:

#1 - script in 'Text Appears When' set to return TRUE if "Job" variable on PC set to 4 like so:

```
int StartingConditional()
{
int nJob = GetLocalInt(GetPCSpeaker(), "Job") == 4;
```

```
return nJob;
}
```

#2 - script in 'Text Appears When' set to return TRUE if "Job" variable on PC set to 3 (set below). If the PC has finished the job, have NPC give reward and set "Job" variable to 4 (no more jobs) in 'Actions Taken' like so:

```
void main()
{
SetLocalInt(GetPCSpeaker(), "Job", 4);
}
```

#3 - script in 'Text Appears When' set to return TRUE if "Job" variable on PC set to 2 (set below). If the PC has finished the job, have NPC give reward and job #3... set "Job" variable to 3 in 'Actions Taken'.

#4 - script in 'Text Appears When' set to return TRUE if "Job" variable on PC set to 1 (set below). If the PC has finished the job, have NPC give reward and job #2... set "Job" variable to 2 in 'Actions Taken'.

#5 - Script in 'Text Appears When' returns TRUE if "TalkedToJoe" variable on the PC set to 1 (set below). In the dialogue, if the PC accepts job#1, set the "Job" variable on the PC to 1 in 'Actions Taken'.

#6 - no script in 'Text Appears When' (if all other scripts above are FALSE, this is the first time the PC has spoken to this NPC). On the first line, put a script in 'Actions Taken' that sets a "TalkedToJoe" variable on the PC to 1.

So this is the basic setting and getting of variables. Storing and using string and object data and other sorts is a little more complicated... but something that is easily caught onto once you get used to manipulating integers. Hopefully this will get you started.

DEFINING AN OBJECT

Now, this topic is a little more complicated... but I've found myself running into this wall more than once when trying to do something.

Most actions require an object as the target. The easiest way to define an object is to have its particular tag handy... which is great if you know what the target object is going to be. But what if you don't? What if you want to find the nearest PC character... whoever it might be? What if you want to target the nearest creature? What if you want to check how far away something is?

There are lots of different way of checking for things like this... I'm going to focus on three, here: `GetNearestCreature`, `GetNearestObject` and `GetDistanceBetween`.

GetNearestCreature

This command allows you to sort through creatures by their nearness to the object by up to three parameters. Note that the target does not have to be the first nearest of that type... that's simply the default. If you wanted to locate the second-nearest non-PC human rogue, for instance, this is the command that you would use.

The grammar is as follows:

```
object GetNearestCreature (int nFirstCreatureType, int  
nFirstParameter, object oTarget=OBJECT_SELF, int nNth=1, int  
nSecondCreatureType = -1, int nSecondParameter = -1, int  
nThirdCreatureType = -1, int nThirdParameter = -1 )
```

This may look a bit confusing, but let's go through it step by step.

The 'nFirstCreatureType' and 'nFirstParameter' are the required initial quality that you are looking for. Say you were looking only for the nearest human... the 'nFirstCreatureType' would be the attribute `CREATURE_TYPE_RACIAL_TYPE`, and the 'nFirstParameter' would be `RACIAL_TYPE_HUMAN`.

Make sense? You have up to three qualities that you can narrow your search by... each requiring both the attribute and the parameter.

The 'oTarget' object is, of course, what the creature you're looking for is near to. It might be `OBJECT_SELF` (and that's the default) if you're looking for the nearness to the one running the script. You could also do something funky like check for the nearest creature to the nearest PC... but I don't even want to type that out.

The integer 'nNth' is what you use to determine whether you want the first-nearest or otherwise. The default is the first nearest... but any number you specify here will determine how many nearer creatures of the parameter you specify are skipped. Note that any function can be used here so long as it returns an integer... you could put `d4()` here, for instance, if you randomly wanted the first- to fourth-nearest.

Here, then, are the list of creature types you can sort by:

CREATURE_TYPE_CLASS = character or monster class

CREATURE_TYPE_DOES_NOT_HAVE_SPELL_EFFECT = if you were looking for a particular creature that didn't have a specific spell on him (referenced by the constant SPELL_*)... I don't imagine this comes up often.

CREATURE_TYPE_SPELL_EFFECT = the opposite of the above, of course... looking for a creature with a particular spell on him. Note that not all spell effects are 'spells', per se.

CREATURE_TYPE_IS_ALIVE = self-explanatory, should you wish to skip the dead ones.

CREATURE_TYPE_PERCEPTION = this is an odd one, as it technically isn't an attribute of the creature but rather the nature of their perception to the target object... like whether they can be seen or heard.

CREATURE_TYPE_PLAYER_CHAR = this basically sorts whether a creature is a PC or not.

CREATURE_TYPE_RACIAL_TYPE = creature race

CREATURE_TYPE_REPUTATION = sorts by faction reputation level to the target.

I won't go through all the particular classes, races and effects that can be used as parameters since they are pretty obvious. Some of the others are not so obvious, however.

```
CREATURE_TYPE_IS_ALIVE:  
TRUE or FALSE (easy)
```

```
CREATURE_TYPE_PERCEPTION:  
PERCEPTION_HEARD  
PERCEPTION_HEARD_ AND_NOT_SEEN  
PERCEPTION_NOT_HEARD  
PERCEPTION_NOT_SEEN  
PERCEPTION_NOT_SEEN_ AND_NOT_HEARD  
PERCEPTION_SEEN  
PERCEPTION_SEEN_AND_HEARD  
PERCEPTION_SEEN_ AND_NOT_HEARD
```

```
CREATURE_TYPE_PLAYER_CHAR:  
PLAYER_CHAR_IS_PC  
PLAYER_CHAR_NOT_PC
```

```
CREATURE_TYPE_REPUTATION:  
REPUTATION_TYPE_ENEMY  
REPUTATION_TYPE_FRIEND  
REPUTATION_TYPE_NEUTRAL
```

So, then... how do we find the nearest PC character to myself?

```
GetNearestCreature (CREATURE_TYPE_PLAYER_CHAR,  
PLAYER_CHAR_IS_PC);
```

The nearest enemy wizard to myself?

```
GetNearestCreature (CREATURE_TYPE_ REPUTATION, REPUTATION_ENEMY,  
OBJECT_SELF, 1, CREATURE_TYPE_CLASS, CLASS_TYPE_WIZARD);
```

The nearest living enemy to my nearest PC ally?

```
GetNearestCreature (CREATURE_TYPE_ REPUTATION, REPUTATION_ENEMY,  
GetNearestCreature (CREATURE_TYPE_ PLAYER_CHAR,  
PLAYER_CHAR_IS_PC, OBJECT_SELF, 1, CREATURE_TYPE_ REPUTATION,  
REPUTATION_FRIENDLY), 1, CREATURE_TYPE_ IS_ALIVE, TRUE);
```

The nearest non-PC that I can see?

```
GetNearestCreature (CREATURE_TYPE_ PLAYER_CHAR,  
PLAYER_CHAR_NOT_PC, OBJECT_SELF, 1, CREATURE_TYPE_ PERCEPTION,  
PERCEPTION_TYPE_ SEEN);
```

GetNearestObject

This is much less complicated simply because you can only sort objects by their type... you don't have the multiple parameters that creatures do. Otherwise, however, it works much the same:

```
object GetNearestObject (int nObjectType=OBJECT_TYPE_ALL, object  
oTarget=OBJECT_SELF, int nNth=1)
```

The 'nObjectType' is, obviously, the kind of object you're looking for. If you don't specify anything, than it will be the nearest object of any kind.

Again, the 'oTarget' object is what your checking the nearest object in relation to... default is OBJECT_SELF, though it doesn't have to be.

'nNth' is the integer that allows you to go beyond the 'nearest'... an nNth of 3 would look for the 'third nearest', for example.

The object types, then, are:

```
OBJECT_TYPE_ALL  
OBJECT_TYPE_AREA_OF_EFFECT  
OBJECT_TYPE_CREATURE
```

OBJECT_TYPE_DOOR
OBJECT_TYPE_INVALID
OBJECT_TYPE_ITEM
OBJECT_TYPE_PLACEABLE
OBJECT_TYPE_STORE
OBJECT_TYPE_TRIGGER
OBJECT_TYPE_WAYPOINT

GetDistanceBetween

This reports the distance (in meters) between two valid objects. It is important that both objects be valid, else the distance returned will be 0.

***float** GetDistanceBetween(object oObjectA, object oObjectB)*

(what's a float, incidentally? A float, like an integer, is just a number... but a float always has decimal places. '50' would be an integer... '50.0' would be a float. If you need to, you can use the command `int FloatToInt(float fFloat)` to convert a float to an integer.)

If you're simply checking how far something is away from the creature or object that is running the script, it is actually easier to use this command:

***float** GetDistanceToObject(object oObject)*

That way you don't have to constantly specify `OBJECT_SELF` in the `GetDistanceBetween` command.

[Edited By David Gaider: Saturday, 06 July 04:48AM (GMT)]

CREATING OBJECTS

```
object CreateItemOnObject(string sItemTemplate, object oTarget =  
OBJECT_SELF, int nStackSize = 1)
```

This creates an item (with the tag of the 'sItemTemplate') in the oTarget object's inventory. We use this often when items are 'given' to a PC... or if, say, an NPC is supposed to have an object on them only at certain times.

If I'm in dialogue and the NPC rewards the PC he's talking to with a regular longsword, I would put a script with this line in the 'Actions Taken' section:

```
CreateItemOnObject ("NW_WSWLS001", GetPCSpeaker());
```

If I want to put 12 regular arrows into a chest that has the tag "CHEST05":

```
CreateItemOnObject ("NW_WAMAR001", GetObjectByTag ("CHEST05"),  
12);
```

(NOTE: If you use ActionGiveItem to have an NPC give an inventory item to the PC, that item *must actually exist* in the NPC's inventory in order for the action to complete. ActionGiveItem will transfer the item to the PC's inventory... CreateItemOnObject creates an entirely new one there.)

(2nd NOTE: I mention that the sItemTemplate used in the command is the item's 'tag'... this is true ONLY for the standard items that come with the game. Sorry, my bad: this is all I'm used to using with this command. If you are trying to use the command with custom items, you must use the blueprint resref for the item... not the tag.)

The 'CreateObject' command is a little bit different. It actually creates the object on the map... this is often referred to as 'spawning'. The object could be a creature or an item. It's structure is as follows:

```
object CreateObject (int nObjectType, string sTemplate, location  
lLoc, int bUseAppearAnimation = FALSE)
```

The 'nObjectType' is a constant that defines what class of object is being created. OBJECT_TYPE_CREATURE, for example, or OBJECT_TYPE_ITEM. This comes from the list of Constants in your script editor.

The 'sTemplate' string is its 'blueprint resref'. Yes, I know... almost every other command

uses an object's tag. This one refers to the resref, usually found under the Advanced tab of the object.

NOTE: *The above is a very common misconception! If you use a tag instead of a resref, it will not work!*

The location consists of an area, an xyz map co-ordinate (which, itself, is known as a 'vector') and a facing. It is easiest to use a *GetLocation(object oObject)* command to get all this information from a waypoint or another object (if you use an object such as a creature, then the spawn will occur in the nearest 'valid' location to it). If need be, you can also construct a location using *Location(object oArea, vector vPosition, float fOrientation)*... though that, quite frankly, is often easily avoided.

The default, obviously, is for the standard appearance animation to not be used... you can opt to use it by specifying TRUE for 'bUseAppearAnimation'.

So... to create a bugbear at an already-laid waypoint that has the tag "WAYPOINT1"...

```
location spawn1 = GetLocation (GetWaypointByTag ("WAYPOINT1"));
CreateObject (OBJECT_TYPE_CREATURE, "NW_BUGBEARA", spawn1);
```

To spawn in a wizard (with the tag "wizard01") with the creation effect, facing the nearest PC...

NWScript:

```
void main()
{
    object oWay = GetWaypointByTag("WAYPOINT1");
    vector vPos = GetPosition(oWay);
    vector vPC =
    GetPosition(GetNearestCreatureToLocation(CREATURE_TYPE_PLAYER_CHAR,
    PLAYER_CHAR_IS_PC, GetLocation(oWay)));
    location lPos = Location(GetArea(oWay), vPos, GetFacing(oWay));
    CreateObject(OBJECT_TYPE_CREATURE, "wizard01", lPos, TRUE);
}
```

ONE LAST THING ABOUT TAGS: Remember that they are case-sensitive! If the tag reads on your creature "wizard01" don't refer to it in your scripts as "WIZARD01" or "Wizard01"... it will not acquire the creature object properly and you will get a badger spawned in. Why a badger? This was simply the 'default summon' used during testing in case an invalid object was referenced for a spawn.

[Edited By David Gaider: Monday, 29 July 11:47PM (GMT)]

REGARDING ANIMATIONS

You basically have two choices when adding animations to your creatures in the game. Adding them directly or using the automatic functions in the generic AI.

Adding Animations Directly

The first thing to realize when you are dealing with scripting animations is that not all creatures have all animations. As a rule of thumb, if the creature is a PC race (human, elf, dwarf, half-orc, gnome or halfling), then it will have all the animations. If the creature is a humanoid monster (bugbear, goblin, etc.) then very likely it has most if not all animations. Non-humanoid monsters and especially birds will be very limited in their animations.

The Action Queue

The second thing to know when scripting animations is how to use the action queue. There are a number of scripting commands which start with the word 'Action'... when a creature calls these commands on themselves, it places the action into a queue. It will finish one action completely and then move onto the next in line... up until the point there are no more actions or a `ClearAllActions()` command is issued.

The reason this is important is that there are two main commands that deal with animations: *ActionPlayAnimation* and just *PlayAnimation*. *ActionPlayAnimation* places the command to perform the animation in the queue... *PlayAnimation* tells the creature to do the animation immediately as soon as it is reached in the script, overriding anything else going on in the queue.

If I wanted to script someone to move to a particular waypoint and then meditate for 6 seconds, it would look like this:

NWScript:

```
void main()
{
    object oTarget = GetNearestObjectByTag("WAYPOINT1");
    ActionMoveToObject(oTarget);
    ActionPlayAnimation(ANIMATION_LOOPING_MEDITATE, 1.0, 6.0);
}
```

The creature would then move to the waypoint and wait until he got there before he began playing his meditation animation.

If I wanted to set a variable when he was finished all that, I would also have to add it into the queue. You can do this with the `ActionDoCommand()`. This places a non-Action command into the queue.

NWScript:

```
void main()
{
    object oTarget = GetNearestObjectByTag("WAYPOINT1");
    ActionMoveToObject(oTarget);
    ActionPlayAnimation(ANIMATION_LOOPING_MEDITATE, 1.0, 6.0);
    ActionDoCommand(SetLocalInt OBJECT_SELF, "Done_Meditation", 1);
}
```

If I did the SetLocalInt command without putting it into the queue, then it would fire as soon as that point in the script was reached... probably well before the creature even reached the waypoint.

The two commands for animations are as follows:

void ActionPlayAnimation (int nAnimation, float fSpeed=1.0, float fSeconds=0.0)

- The 'nAnimation' is Constant for the animation being played.
- 'fSpeed' is the speed at which the animation is played... you could have a creature turn its head very slowly or very quickly, for instance... 1.0 is normal speed.
- 'fSeconds' is only used for looping animations (like the meditation)... it determines how long you wish the animation to be played. If left blank on a looping animation, it will play that animation until told to do something else.

void PlayAnimation (int nAnimation, float fSpeed=1.0, float fSeconds=0.0)

As mentioned, this is the same as the ActionPlayAnimation command, except that the animation is not placed in the queue... it is played immediately.

Animation Constants

You can find a list of all the animation constants (used in the 'nAnimation' portion of the command) by selecting the 'Constants' button in your script editor... all the constants begin with ANIMATION_*

There are two types of animations: 'fire-and-forget' (or FNF), which only plays once and no duration is needed, and 'looping' which play as long as needed and a duration is required for.

A reminder once again: NOT ALL MODELS HAVE ALL ANIMATIONS. Just to mention, too, that the animations listed in the constants are not every animation that a model is capable of (there is a dying animation, after all, as well as combat animations and others)... this is just the current list of the ones that can be played via script.

Using Generic AI Animations

For a quick and easy solution to adding some life to your placed creatures, the generic AI has two functions that you can use.

In the generic OnSpawn script ("nw_c2_default9"), there is a whole list of commands that are all commented out (they are preceded with a '/' double slash that colors them green and prevents them from being compiled). To use the built-in animations, you simply need to comment back in (remove the '/') one of the following commands:

NWScript:

```
SetSpawnInCondition(NW_FLAG_AMBIENT_ANIMATIONS);  
  
SetSpawnInCondition(NW_FLAG_IMMOBILE_AMBIENT_ANIMATIONS);
```

Don't worry about the comments that are on the same line with these commands... that just tells what they do. Simply erase the double-slash at the beginning of the line.

Then you re-compile the script and save it as a different file. And that's it... that's all you have to do.

What do these do?

Basically they are called in the OnHeartbeat event (meaning the script will 'activate' every 6 seconds). The script checks to make sure that the creature is not asleep, not in combat, not in conversation and no enemy is in sight... if all those are okay, it plays the animations.

'ambient animations' means that the creature will move about randomly, occasionally stopping to turn to nearby friends (creatures with a friendly reputation) and play what social animations it has (and, yes, this will work on any type of creature.. it will do what it can for those creatures who don't have the full range).

'immobile ambient animation' does the same thing... without the random movement. The creature stays in place.

So you can put down several of these types of creatures, for instance, and they will turn to each other at random intervals and seem to chat, laugh, argue... and even mill around and mingle, with the ambient animations.

Do the placeable object animations work the same way?

Yes. You can tell a chest to open by having it run its ANIMATION_PLACEABLE_OPEN, or a lamp post to turn off using ANIMATION_PLACEABLE_DEACTIVATE. A few things to keep in mind:

1) For placeable objects that are sources of illumination (such as the lamp post), it is not enough to just use its ANIMATION_PLACEABLE_DEACTIVATE or ANIMATION_PLACEABLE_ACTIVATE. That just affects the glowing part of the animation, itself. You must also use the SetPlaceableIllumination command set to TRUE and tell the area it's in to RecomputeStaticLighting.

The following is an example of placeable illumination use:

NWScript:

```
// will turn the lightable object on and off when selected
// placed in its OnUsed event
void main()
{
    if (GetLocalInt (OBJECT_SELF,"NW_L_AMION") == 0)
    {
        SetLocalInt (OBJECT_SELF,"NW_L_AMION",1);
        PlayAnimation (ANIMATION_PLACEABLE_ACTIVATE);
        SetPlaceableIllumination (OBJECT_SELF, TRUE);
        RecomputeStaticLighting (GetArea(OBJECT_SELF));
    }
    else
    {
        SetLocalInt (OBJECT_SELF,"NW_L_AMION",0);
        PlayAnimation (ANIMATION_PLACEABLE_DEACTIVATE);
        SetPlaceableIllumination (OBJECT_SELF, FALSE);
        RecomputeStaticLighting (GetArea(OBJECT_SELF));
    }
}
```

2) Doors are not placeable objects. First thing you should know about them is that if the door is unlocked, a creature who is told to move to a point on the other side of one will automatically open it.

Beyond that, the commands for doors are as follows:

- ActionOpenDoor: If used in the script of a creature, it will move to the door and open it (if it is unlocked). If used in the script of the door (or the command is sent to the door object via AssignCommand), then the door will open itself.
- ActionCloseDoor: As above, only the door is closed.
- ActionLockObject: If used in the script of a creature, it will move to the object (can be a door or placeable) and attempt to use its Open Locks skill to unlock it. **ONLY call it in the script of a creature!**
- ActionUnlockObject: As above, except the door or object is unlocked.
- SetLocked: This is the command you use if you want a door or object to be set to locked

or unlocked without the aid of a creature or skill. If 'bLocked' is set to TRUE, the object will be locked... if set to FALSE, it will be unlocked. (example: if called in a door's own script, SetLocked (OBJECT_SELF, TRUE) will cause it to be locked.)

NWScript:

```
// set in a door's OnHeartbeat script, this will cause
// it to close and lock itself at dusk
// and unlock itself at dawn
void main()
{
    if (GetIsDusk() && GetIsOpen (OBJECT_SELF))
    {
        ActionCloseDoor (OBJECT_SELF)
        // SetLocked is set in an ActionDoCommand because we
        // want it to be in the door's queue... we want the
        // ActionCloseDoor to be completed before locking the door
        ActionDoCommand (SetLocked (OBJECT_SELF, TRUE))
    }
    else if (GetIsDawn() && GetLocked (OBJECT_SELF))
    {
        SetLocked (OBJECT_SELF, FALSE);
    }
}
```

EXPLAINING WAYPOINTS AND WALKWAYPOINTS

A *waypoint* is an invisible object that is placed in an area to mark a specific location. It has its own tag (which you can set when it is created) and can optionally be made into a 'map note' (one could be labeled 'General Store', for instance, and in the mini-map an icon would appear where it was located... if the player moved his cursor over that icon it would display its label).

The generic AI that comes with NWN is set up so that you can get creatures you set down to walk a circuit of these waypoints automatically.

How is this done? It all has to do with the tag you assign to the waypoint. Take the tag of the creature(s) you want to use the waypoint... add the prefix "WP_" and the suffix "_0X"... and the creature will automatically begin walking them in that order.

Example:

I have an NPC guard who I have given the tag "GUARD". I lay down four waypoints which form his patrol circuit. I give them the tags "WP_GUARD_01", "WP_GUARD_02", "WP_GUARD_03" and "WP_GUARD_04".

According to the default AI, when the game started he would proceed to the nearest waypoint (whether it is "WP_GUARD_01" or not) and then move to the next waypoint in numerical order. He keeps going, in order, pausing 1 second at each one. Once he got to "WP_GUARD_04" he would start back to "WP_GUARD_01". If he saw anything hostile, he would move to attack... and, once the combat was over, he would return to his patrol.

The above is the simplest use of the waypoints, of course... there are many other ways to tweak it. There are likely a few questions right off the bat that might arise:

1. If I used a lot of waypoints for my NPC's, there could be a LOT of waypoints on the screen!

One waypoint doesn't only have to serve one creature... it serves one creature tag. More than one creature can have the same tag, if you wish.

There are also four different colored appearances that you can set on the waypoints to distinguish them from each other.

Even so, you could have quite a few in any given area, depending on what you were doing. Fortunately waypoints don't run scripts or take up any graphics processing power by their existence. If they get in the way of you seeing things in the toolset, there is a button which allows you to screen them out of view (there are buttons for other things, too, if you don't wish to view objects, creatures, items or whatever).

2. *It sounds like a lot of work to re-name all the tags on those waypoints.*

You could edit each waypoint and change its tag if you wanted to. Fortunately you can hold down the SHIFT key and select the waypoints in the order you want them... then right-click the mouse button and select 'Create Set'. If you provide the tag name ('WP_GUARD' in the above example), it will automatically change their tags and add on the numbered suffixes for you.

3. *What if I want part of the circuit to cross an area transition... like going into a house or something?*

This works fine. I suspect you would have to have a regular area transition created by the wizard (the ones in triggers and doorways), however... not a transition that you scripted yourself.

I recently creating a peasant who walked four waypoints, two of them being in another area (inside a house). I then planted a house tile in the main area and connected the two with a door. He left the door open after he used it the first time, but otherwise it worked fine.

More Advanced Uses

Posts: Instead of walking waypoints, you can assign a creature to a 'post'. Lay down a single waypoint with the prefix 'POST_' added to the creature's name.

The creature will then remain at his post. If he moves from it to engage in combat, he will return to it as soon as he is done.

Day/Night Activities: In the generic OnSpawn script, you can uncomment a command called 'SetSpawnInCondition (NW_FLAG_DAY_NIGHT_POSTING)'. This will tell the creature to differentiate between night and day.

How does this affect waypoints? Well, you can give a creature a different set of waypoints to walk (or a different post) in the day and night by changing the prefix of your waypoints:

```
day waypoint: prefix "WP_"
night waypoint: prefix "WN_"
day post: prefix "POST_"
night post: prefix "NIGHT_"
```

So you could set up the guard in the example to walk his waypoints during the day. Then, if that flag was uncommented in the OnSpawn script, you could set a "NIGHT_GUARD" waypoint in his barracks that he would return to that spot when it was dark.

One thing to remember is that when you make any changes to the OnSpawn script, make sure you save it under a different name before compiling... else you'll make a global change that will affect every creature in your module.

THE USERDEFINED EVENT IS YOUR FRIEND

Seriously. No joke at all here. There will naturally be some users who will prefer to go deeper and customize the various scripts that make up the generic AI... some will even go further than that and come up with their own versions of the generic AI include file. That's great, and there's nothing wrong with doing that.

The great thing is: *you don't have to*. 98% of the creature scripting that was done in the NWN official campaign was done using UserDefinedEvents. I sat down with Preston (one of our scripters and one of the main guys behind the generic AI) and fired off a rapid series of things I would like to script... just about every one was do-able using the UserDefinedEvent. And it is easy to implement.

Let's back up for a second here and discuss what an event is. Basically there are numerous scripts (sets of instructions) that exist on a creature... and each one is tied to events. When an event occurs, it tells the creature to fire off the proper script. No event = the script doesn't run.

There are a number of events which get mentioned here a lot. The 'OnHeartbeat' event fires every 6 seconds... so the script attached to that even runs that often. The 'OnPerceived' event fires whenever the NPC perceives anything... the script attached gets run. These scripts can even fire off other events, prompting different scripts to be run... and therein lies the UserDefinedEvent.

Basically, the UserDefinedEvent allows you to access the normal events a creature uses (OnPerception, OnDeath, etc.) without disturbing the normal AI routines or modifying the already-existing scripts.

Using the UserDefinedEvent

There is one script on every creature that has been mentioned lots before, and it's one you will get to know well. This is the OnSpawn script. This is an event that runs only once when the creature first comes into existence... think of it as an 'initializing' script.

The generic OnSpawn script has a whole series of commands in it which are commented out... as a matter of fact, without the user changing anything in it, all that large script really does is three commands: it tells the creature to walk waypoints that refer specifically to its tag, set up its listening patterns (so it will respond to shouts from other NPC regarding combat) and generate a small amount of generic treasure in inventory (some gold, etc.)

It is designed to allow the user to set states that access the generic AI, as they wish. All

they do is go through the list and uncomment any state flags they wish to use (remove the '/' double slashes at the beginning of the command). Then save the OnSpawn script under a different name and use it in place of the regular one... voila, the generic behaviors have been altered.

The state flags we need to pay attention to are these:

```
SetSpawnInCondition (NW_FLAG_PERCIEVE_EVENT);  
//OPTIONAL BEHAVIOR - Fire User Defined Event 1002  
  
SetSpawnInCondition (NW_FLAG_ATTACK_EVENT);  
//OPTIONAL BEHAVIOR - Fire User Defined Event 1005  
  
SetSpawnInCondition (NW_FLAG_DAMAGED_EVENT);  
//OPTIONAL BEHAVIOR - Fire User Defined Event 1006  
  
SetSpawnInCondition (NW_FLAG_DISTURBED_EVENT);  
//OPTIONAL BEHAVIOR - Fire User Defined Event 1008  
  
SetSpawnInCondition (NW_FLAG_END_COMBAT_ROUND_EVENT);  
//OPTIONAL BEHAVIOR - Fire User Defined Event 1003  
  
SetSpawnInCondition (NW_FLAG_ON_DIALOGUE_EVENT);  
//OPTIONAL BEHAVIOR - Fire User Defined Event 1004  
  
SetSpawnInCondition (NW_FLAG_DEATH_EVENT);  
//OPTIONAL BEHAVIOR - Fire User Defined Event 1007
```

What Next?

Okay... you've uncommented the flags you want to script for and re-saved the OnSpawn script. That's taken you 2 minutes. Now how do you use it?

Every creature has an OnUserDefined event section that you can put a script in. For any creature you make, it will be completely blank... this is where you do your actual scripting. Any script you put here will be 'checked' every time an event of the type you uncommented occurs. So if you uncomment the 'SetSpawnInCondition (NW_FLAG_DISTURBED_EVENT)', then every time the creature's inventory was disturbed (say someone pickpockets an item), the OnUserDefined script would also be run.

The basic OnUserDefined script would look similar to this:

NWScript:

```
void main()  
{  
    int nUser = GetUserDefinedEventNumber();
```

```

        if (nUser == the number in the OnSpawn, from 1001 to 1007)
        {
            (do something)
        }
    }
}

```

Let's say I want to make a creature wave to the first PC that he sees. I go into the OnSpawn and un-comment 'SetSpawnInCondition (NW_FLAG_PERCIEVE_EVENT)' and then re-save it as a new script.

Then I create a script for the OnUserDefined perhaps as so:

NWScript:

```

// this script will make the NPC wave to a PC upon perceiving
// them
// remember that if the PC starts off the module in the NPC's
// perception
// range, no event will fire (because perception has not changed)
void main()
{
    int nUser = GetUserDefinedEventNumber();
    if (nUser == 1002) // OnPerception event
    {
        object oCreature = GetLastPerceived();
        int nWave = GetLocalInt(OBJECT_SELF, "Wave_Once");
        //check to see if I actually saw a PC
        if (GetLastPerceptionSeen() && GetIsPC(oCreature) && (nWave
== 0))
        {
            // wave only once
            SetLocalInt(OBJECT_SELF, "Wave_Once", 1);
            // set a timer so that the NPC can wave again 15 seconds
            later
            DelayCommand(15.0, SetLocalInt(OBJECT_SELF, "Wave_Once",
0));
            // turn to face the PC
            ActionDoCommand(SetFacingPoint(GetPosition(oCreature)));
            // wave howdy
            ActionPlayAnimation(ANIMATION_FIREFORGET_GREETING);
            SpeakString("Hello!");
        }
    }
}

```

And that's it. I could also set other checks like a minimum distance, see if the PC is

facing me... or get as complex a behavior as I wish... and I don't have to worry about screwing up the AI scripting that's already sitting in the OnPerceive event section.

USING THE SCRIPT TEMPLATES

by Yaedaïen

A brief note here about an easier way to create new OnSpawn and OnUserDefined scripts by way of using the 'Templates' button in your script editor. Thanks to Yaedaïen for writing this part up.

When you want to change a creature's default OnSpawn script, open the creature's properties, and go to the OnSpawn script. Delete the script name you see there (it should be called "nw_c2_default9"), and type a new name that is meaningful to you (like "MyBalor_Spawn"). Then click the *Edit* button. The toolset will inform you that the script does not exist (which is normal), click OK to create the script. You will then see the following blank script, and you will notice that it has whatever title you typed in the OnSpawn field:

```
void main()  
{  
  
}
```

Delete this, so that there is nothing in your script, then press the (b)Templates] button, found in the pane on the right of the Script Editor.

This will bring up a selection of four items. Since this is an OnSpawn script, double-click the (i)on spawn in script(/i) option and you'll see a brand new, rather long script appear in your editor. This is now your base OnSpawn script, and you can edit it like any other OnSpawn script, except you know that you won't accidentally overwrite anything else that is using the original script.

Next, you would want to create your UserDefined script, so save your new OnSpawn script, and exit the Script Editor.

Move down to the last field on the Properties tab, named OnUserDefined, and replace the script name you find there (it should be called "nw_c2_defaultd") with something that's meaningful to you (like "MyBalor_UserDef"). Just like with OnSpawn, click the *Edit* button, and bring up the Script Editor. Again, just like OnSpawn, delete the initial, blank script, and press the *Templates* button. This time, however, double click the *on user defined script* selection, and then code away.

Remember to save your script, and save your module for your character's new scripts to take effect!

MORE FULLY-COMMENTED ONSPAWN SCRIPT

Not all the lines in the regular OnSpawn script ("nw_c2_default9") explain how their functions are actually used. This is an updated version that you may use if you wish.

NWScript:

```

//:////////////////////
//: Custom On Spawn In
//: File Name
//: Copyright (c) 2002 Bioware Corp.
//:////////////////////
/*
*/
//:////////////////////
//: Created By:
//: Created On:
//:////////////////////
#include "NW_O2_CONINCLUDE"
#include "NW_I0_GENERIC"
void main()
{
// OPTIONAL BEHAVIORS (Comment In or Out to Activate )
*****
//SetSpawnInCondition(NW_FLAG_SPECIAL_CONVERSATION);
// This causes the NPC to speak a single line from their dialogue
file
// upon perceiving a player. Make sure that the line being spoken
is at the
// very top of the NPC's other dialogue starting nodes and that
you have
// placed the script "nw_d2_gen_check" in that line's 'Text
Appears When' area.
// Do not use this flag for hostile creatures.
//SetSpawnInCondition(NW_FLAG_SPECIAL_COMBAT_CONVERSATION);
// This flag is similar to the above... except that it allows a
hostile NPC
// to display a single line of dialogue before attacking. Put the
line into the
// NPC's dialogue file as above, but place "nw_d2_gen_combat" into
'Text Appears When'
//SetSpawnInCondition(NW_FLAG_SHOUT_ATTACK_MY_TARGET);
// This sets up the NPC so that any NPC of a faction who is
friendly to it
// who is attacked or attacks an enemy (and is using the generic
AI) will issue a shout
// that this NPC will now listen and respond to.
//SetSpawnInCondition(NW_FLAG_STEALTH);
// If the NPC has Hide skill they will automatically be in Stealth
Mode
// but only when the WalkWayPoints command is called (below)
//SetSpawnInCondition(NW_FLAG_SEARCH);
// If the NPC has the Search skill they are automatically in
Search Mode
// but only when the WalkWayPoints command is called (below)
//SetSpawnInCondition(NW_FLAG_SET_WARNINGS);
// This will set the NPC to give a warning to non-enemies before
attacking
//SetSpawnInCondition(NW_FLAG_DAY_NIGHT_POSTING);
// This separates the NPC's waypoints into night and day. Normally
a waypoint prefix "WP"
// or "POST" would be used always. If this flag is set, those
prefixes are used in the day
// and "WN" or "NIGHT" prefixes are used at night.
//SetSpawnInCondition(NW_FLAG_APPEAR_SPAWN_IN_ANIMATION);
// when the creature spawns in, it uses EffectAppear() instead of
fading in
// but only if SetListeningPatterns is called (below)
//SetSpawnInCondition(NW_FLAG_IMMOBILE_AMBIENT_ANIMATIONS);

```

```

        // this causes the NPC to use common animations it possesses, and
will appear more
        // social if placed near a friendly NPC (they will turn to each
other and use social animations)
        //SetSpawnInCondition(NW_FLAG_AMBIENT_ANIMATIONS);
        //This is similar to the above, except that the creature will also
move around randomly
        //NOTE that these animations will play automatically for Encounter
Creatures.
        // NOTE: ONLY ONE OF THE FOLOWING ESCAPE COMMANDS SHOULD EVER BE ACTIVATED AT
ANY ONE TIME.
        //SetSpawnInCondition(NW_FLAG_ESCAPE_RETURN); // OPTIONAL BEHAVIOR (Flee to
a way point and return a short time later.)
        //SetSpawnInCondition(NW_FLAG_ESCAPE_LEAVE); // OPTIONAL BEHAVIOR (Flee to
a way point and do not return.)
        //SetSpawnInCondition(NW_FLAG_TELEPORT_LEAVE); // OPTIONAL BEHAVIOR
(Teleport to safety and do not return.)
        //SetSpawnInCondition(NW_FLAG_TELEPORT_RETURN); // OPTIONAL BEHAVIOR
(Teleport to safety and return a short time later.)
        // to use 'escape', you need a waypoint the NPC will flee to with
the tag "EXIT_" + the NPC's tag
        // these commands can be activated in a script by calling
ActivateFleeToExit(), "NW_IO_GENERIC" must be included
        // for the escape commands that have 'return', the NPC will be re-
spawned at it's starting location
// CUSTOM USER DEFINED EVENTS
/*
    The following settings will allow the user to fire one of the blank user
defined events in the NW_D2_DefaultD. Like the
    On Spawn In script this script is meant to be customized by the end user to
allow for unique behaviors. The user defined
    events user 1000 - 1010
*/
        //SetSpawnInCondition(NW_FLAG_HEARTBEAT_EVENT); //OPTIONAL BEHAVIOR -
Fire User Defined Event 1001
        //SetSpawnInCondition(NW_FLAG_PERCIEVE_EVENT); //OPTIONAL BEHAVIOR -
Fire User Defined Event 1002
        //SetSpawnInCondition(NW_FLAG_ATTACK_EVENT); //OPTIONAL BEHAVIOR -
Fire User Defined Event 1005
        //SetSpawnInCondition(NW_FLAG_DAMAGED_EVENT); //OPTIONAL BEHAVIOR -
Fire User Defined Event 1006
        //SetSpawnInCondition(NW_FLAG_DISTURBED_EVENT); //OPTIONAL BEHAVIOR -
Fire User Defined Event 1008
        //SetSpawnInCondition(NW_FLAG_END_COMBAT_ROUND_EVENT); //OPTIONAL BEHAVIOR -
Fire User Defined Event 1003
        //SetSpawnInCondition(NW_FLAG_ON_DIALOGUE_EVENT); //OPTIONAL BEHAVIOR -
Fire User Defined Event 1004
        //SetSpawnInCondition(NW_FLAG_DEATH_EVENT); //OPTIONAL BEHAVIOR -
Fire User Defined Event 1007
// DEFAULT GENERIC BEHAVIOR (DO NOT TOUCH)
*****
        SetListeningPatterns(); // Goes through and sets up which shouts the NPC
will listen to.
        WalkWayPoints(); // Optional Parameter: void WalkWayPoints(int nRun
= FALSE, float fPause = 1.0)
        // 1. Looks to see if any Way Points in the module
have the tag "WP_" + NPC TAG + "_0X", if so walk them
        // 2. If the tag of the Way Point is "POST_" + NPC
TAG the creature will return this way point after
        // combat.
        // To get the NPC to continue walking waypoints
after conversation, you must go
        // into its dialogue file and, under 'Other Files',
call this command again in the
        // 'End' and 'Aborted' script sections
        GenerateNPCTreasure(); // * Use this to create a small amount of treasure
on the creature
    }

```


REWARDING GOLD AND XP

The two basic commands for doing this are very, very simple.

void GiveGoldToCreature(object oCreature, int nGP)

void GiveXPToCreature(object oCreature, int nXpAmount)

It doesn't matter where the gold or XP is coming from. just where it is going to... so these commands can be run from anywhere.

To give 100 gold to the PC I'm speaking to (placed in a script within the 'Actions Taken'):

GiveGoldToCreature (GetPCSpeaker(), 100);

To give 100 XP to each member of the party, starting with the nearest one to me:

NWScript:

```
object oFirstMember =
GetNearestCreature(CREATURE_TYPE_PLAYER_CHAR, PLAYER_CHAR_IS_PC);
object oPartyMember = GetFirstFactionMember(oFirstMember, TRUE);
while (GetIsObjectValid(oPartyMember) == TRUE)
{
    GiveXPToCreature(oPartyMember, 100);
    oPartyMember = GetNextFactionMember(oFirstMember, TRUE);
}
}
```

(personally, I would put that into a function so I wouldn't have to type it all out every time)

Some other stuff that is handy to know with regards to experience and gold:

- When you write journals in the journal editor, you can assign an 'XP Value' to each entry. This number can be retrieved by the following command:

int GetJournalQuestExperience(string szPlotID)

In the campaign, we have a PlotID for each quest (which can consist of several journal entries right up to 'completion') and each PlotID has an XP value. Then we dole out the XP in percentages as the plot gets completed. 50% of the total for this, 25% for that, etc.

- How much gold does someone have?

```
int GetGold( object oTarget = OBJECT_SELF)
```

So to find out how much gold is on the person running the script you just go GetGold()... but it literally could target anyone.

- if you want someone to take gold from someone else, you use this:

```
void TakeGoldFromCreature( int nAmount, object  
oCreatureToTakeFrom, int bDestroy = FALSE)
```

If 'bDestroy' is marked as TRUE, then the creature taking the gold doesn't end up with it in their inventory... the gold is destroyed and removed from the world.

MAKING SKILL CHECKS

First off, if you don't know how D&D 3rd edition is set up for skill checks, here's the basic of how it goes: whenever you're trying to perform an action using your skills, you roll the d20 (20-sided die, getting a number from 1 to 20) and add to that both the rank in the skill and the ability bonus for whatever ability is tied to that skill. Higher is always better, and you are trying to equal or exceed the DC... the 'difficulty class' number which is set by the DM.

So let's say a player wants to use his Disable Traps skill on a trap he's found. The DM determines the DC to be 25 (a fairly hard task). The player has 8 ranks in Disable Traps and a +3 modifier to his Dexterity (the ability tied to Disable Traps). He rolls the d20 and adds +11 to his roll, needing a 14 or higher to succeed in the task.

For such things as Disable Traps and most other skills, the skill checks are handled by the game engine and often all that is needed from the DM is the DC (if that).

There is no automatic command in NWScript that does these checks for you, however, if you are using a skill check in a script (whether to check for success of Persuade in dialogue or some other use of a skill that isn't automatically covered). If you want to use them, you're going to have to either script it or use the function in the official campaign that we've made.

the Official Campaign's function

The thing about the AutoDC command that's defined in the official campaign that some might not like is that the DC is automatically generated... it is not specified each time.

When the scripter uses the command, they specify either DC_EASY, DC_MEDIUM or DC_HARD and a DC is generated based on the level of the NPC.

The syntax of it is as follows:

```
int AutoDC(int DC, int nSkill, object oTarget)
```

So if I wanted to make a persuade of an NPC that the PC is in dialogue with a hard test, I could use this:

```
AutoDC(DC_HARD, SKILL_PERSUADE, GetPCSpeaker());
```

A note here that a skill check returns an integer (the 'int') because the command should return either TRUE or FALSE. False (or a '0' integer) means that the check has failed. True (or a '1' integer) means that the check has succeeded.

If you're using a skill check in the dialogue editor, you need to 'split' the conversation... meaning that more than one dialogue stems from the same PC response. When the dialogue is presented with choices like this, it looks to the scripts in 'Text Appears When', starting with the top node and working its way down to the bottom. The first node that

has a script that returns TRUE (or has no script), it uses.

An example of a script with an easy Persuade check:

NWScript:

```
#include "NW_I0_PLOT"  
int StartingConditional()  
{  
int iCheck = AutoDC(DC_HARD, SKILL_PERSUADE, GetPCSpeaker()) ==  
TRUE;  
return iCheck;  
}
```

So you could have something like this:

"Oh, I certainly couldn't tell you that. It's confidential."

A. <Persuade> "Oh, I'm sure there's no harm in telling me."

B. <Threaten> "Tell me or suffer the consequences, wench."

C. "Let me ask you something else, then."

If I selected option B, the dialogue would proceed on two paths:

|
|

--> #1. (with persuade script in 'Text Appears When') "I suppose you're right. Ok, this is how it is..."

|
|

--> #2. (with no script) "Forget it, I'm not telling you."

#1 above is the 'successful' path, #2 the 'failed' path. If the script in #1 returns TRUE then that path is followed and the player was successful. No script is needed on the failed path... if the successful path was FALSE, that meant the PC failed his skill check.

Writing Your Own Function

To write your own skill checks, you just follow the same logic that you would as if you were about to do the check in pen-and-paper. You need the PC's bonus to the die roll, the die roll and the DC (provided by the DM).

Some commands you might want to use:

```
int GetSkillRank(int nSkill, object oTarget=OBJECT_SELF)
```

This returns the rank + ability bonus of the oTarget's skill.

```
int GetLevelByClass(int nClassType, object oCreature =  
OBJECT_SELF)
```

This returns the creature's level in a particular class (such as CLASS_TYPE_FIGHTER).

```
int GetHitDice(object oCreature)
```

This is useful if you want the character level of the PC, regardless of their class(es).

```
int GetAbilityModifier(int nAbility, object  
oCreature=OBJECT_SELF)
```

Gets the modifier for a particular ability, useful if you want a strength check only, or something similar.

At any rate, here's an example of a quick home-made skill check, say using the Heal skill on a dialogue node:

NWScript:

```
int StartingConditional()  
{  
    int iDC = 20; // or whatever the DM wishes to set it to  
    int iBonus = GetSkillRank(SKILL_HEAL, GetPCSpeaker());  
    if ((d20() + iBonus) >= iDC)  
    {  
        return TRUE;  
    }  
    return FALSE;  
}
```

Of course this is just an example... there is a built-in method for using Heal normally (with the ActionUseSkill command, actually)... this kind of scripting would be for custom skill checks, which I know people will want to do.

MAKING UNIQUE ITEMS

How does one script their own items that go beyond the standard item effects... how do you make an effect conditional, for instance? How would I script a Wand of Wonder or a talking sword?

Well, the first thing to realize is that, unlike other objects and creatures in the game, items do not have their own scripts. You do not pull up an item's properties and specify which scripts it uses... it can't use any.

The way that you can script things that items do has to be on the module level. In the Module Properties, you have three 'events' which affect items. When these events occur, it prompts the module to run whatever scripts are tied to these items.

Two of these events are **OnAcquireItem** and **OnUnAcquireItem**. This means every time an item is picked up or dropped, an event is 'fired' which starts the appropriate script. If no script is there, nothing happens.

So if I want to set a certain variable when a particular sword (that has the tag "SWORD01") is picked up, say, I could put this script in the OnAcquireItem section of the module:

NWScript:

```
void main ()
{
    object oItem = GetObjectByTag("SWORD01");
    int iVar = GetLocalInt(GetItemPossessor(oItem),
"VARIABLENAME");
    // below is 'if the firer of the event is the one I've
specified
    // and the variable hasn't been set, set the variable on the
// item's possessor'
    if ((GetModuleItemAcquired() == oItem) && (iVar == 0))
    {
        SetLocalInt(GetItemPossessor(oItem), "VARIABLENAME", 1);
    }
}
```

That doesn't do much for scripting your own item properties, however, does it? For that, you need to take advantage of the third module event: **OnActivateItem()**.

Activating an item requires a specific 'activation' event. When you make an item, you can assign to it a 'Cast Spell' property... and two of the choices will be *Unique Power* and *Unique Power: Self Only*. The difference between these is that the second simply triggers the 'OnActivateItem' event... the first requires you to target the Unique Power, first.

This 'Unique Power' shows up on the radial menu of the item just as any other ability would. If you wanted to specify what that ability was to the user, you could include it in the item's description (of which there is an 'identified' as well as an 'unidentified' description, should you wish to use both).

When writing the script to place in `OnActivateItem()`, there are only a few references which you will need to keep in mind:

object `GetItemActivated()`

This returns the item that caused the last `OnActivateItem()` event.

object `GetItemActivatedTarget()`

This returns the target of the Unique Power, if it was targeted at a specific object.

location `GetItemActivatedTargetLocation()`

This returns the target of the Unique Power, if it was targeted at a location rather than a specific object.

object `GetItemActivator()`

This returns the creature object of whoever activated the item... a bit handier than going through the whole `GetItemPossessedBy` rigmarole.

And that's all you need. With these, you can script as many activated properties for as many items as you wish. Keep in mind, of course, that these scripts are module-specific. If you take the item to another module that does not have that script (or another that looks for that item) in their `OnActivateItem()`, then nothing will happen.

NOTE: there is currently a bug with `OnActivateItem` with regards to single-use items (and this also applies to items on their last charge). When an item is used up, it disappears... the event still fires, but all of the commands such as `GetItemActivated` and such return an invalid object (because the item in question no longer exists).

SITTING IN CHAIRS AND SLEEPING

How do I get my PC to sit on a chair?

There is currently no emote to sit in a chair. Until one is added for PC characters to use, the script to make a PC sit down in a chair has to be on the object, itself.

If you put down a useable placeable object chair (the chair has the Useable box checked, and make sure the PC will have room to maneuver while sitting), put this script in the OnUsed event in the Scripts tab:

NWScript:

```
void main()
{
    object oChair = OBJECT_SELF;
    if(!GetIsObjectValid(GetSittingCreature(oChair)))
    {
        AssignCommand(GetLastUsedBy(), ActionSit(oChair));
    }
}
```

When I place an NPC in the toolset, they just stand there. Is there no way I can have them sitting down?

Not in the toolset, no... the creature must be standing. The creature would have to be scripted so that when the game begins it finds a chair and performs the ActionSit command.

Create a placeable object chair and give it the tag "CHAIR". Place the NPC next to the chair. Copy and save their OnSpawn script under another name, and add this line at the bottom:

```
ActionSit (GetNearestObjectByTag ("CHAIR", OBJECT_SELF));
```

Note that when a PC speaks the NPC, they will stand up. To get them to sit back down, you will need to go to the 'Other Files' tab in their dialogue. You will see spots for two scripts there, which activate when dialogue is either ended or aborted. Make a script that calls the above line and they will sit back down once dialogue is over.

This is all fine and well, but what if I want the NPC to sit on the chairs already in the tileset?

Some users have reported success by following the same process as sitting in a chair... except find the 'invisible object' placeable object and place that on the seat of the chair.

Give that object the tag "CHAIR" and use the same ActionSit line as above and they should sit down for you just the same.

What if I want my NPC sleeping on a bed?

There is no way currently to actually get a creature into a placeable object bed or a tileset bed. A placeable object body can be placed onto a bed, but not a creature.

A creature can, however, sleep on a placeable object bedroll or on the floor. If you wanted your NPC to start off the game sleeping, you need to copy the creature's OnSpawn script, save it under a different name, and then add this to the bottom of that script:

```
effect eLieDown = EffectSleep();
effect eSnore = EffectVisualEffect (VFX_IMP_SLEEP);
effect eSleep = EffectLinkEffects (eLieDown, eSnore);
ApplyEffectToObject (DURATION_TYPE_PERMANENT, eSleep,
OBJECT_SELF);
```

Waking Up Your NPC

If you want the NPC to wake up at some point, you must use the RemoveEffect command. An effect, once applied, cannot simply have another variable defined as EffectSleep, however, and removed by the command. The 'pointer' must be on that particular effect. If the effect is the same variable in the same script as when it was applied, you could simply use this:

```
RemoveEffect (OBJECT_SELF, eSleep);
```

Normally, however, you must scan through an object's effects and find the one you're looking for through the effect type. The following is a simple script to remove an object's sleep effect (this one in the OnHeartbeat of the creature should any hostile come within 5 meters):

NWScript:

```
// wake-up script
#include "NW_I0_GENERIC"
void main()
{
    // if I am asleep
    if (GetHasEffect(EFFECT_TYPE_SLEEP))
    {
        // get the nearest enemy creature to me
        object oTarget =
        GetNearestCreature(CREATURE_TYPE_REPUTATION,
        REPUTATION_TYPE_ENEMY);
        // and if there is one and it is less than 5 meters away
```

```

        if ((GetDistanceToObject(oTarget) < 5.0) &&
(GetIsObjectValid(oTarget)))
        {
            effect eSleep = GetFirstEffect(OBJECT_SELF);
            // scroll through my current effects
            while (GetIsEffectValid(eSleep))
            {
                // and if one of them if the effect sleep but
didn't come from a sleep spell
                if ((GetEffectType(eSleep) == EFFECT_TYPE_SLEEP)
&& (GetEffectSpellId(eSleep) != SPELL_SLEEP))
                {
                    // remove it
                    RemoveEffect(OBJECT_SELF, eSleep);
                }
                eSleep = GetNextEffect(OBJECT_SELF);
            }
        }
    }
}

```

HOW DO I MAKE MY NPC'S INITIATE DIALOGUE ON THEIR OWN?

OK, first off: until you know EXACTLY what you are doing, do NOT alter the script that is normally in the OnPerception event ("nw_c2_default2"). Leave it alone. If anyone tells you to alter anything but the OnSpawn and OnUserDefined scripts for you NPCs and you aren't confident of doing so, tell them they're full of monkeypoo.

As mentioned earlier, here, regarding UserDefined events, the generic AI is set up to allow you to use these events without altering the default scripting.

Using the OnPerceive method of starting conversation

1) Go into your OnSpawn script. Save it under a new name. Then go down to the line where it says 'SetSpawnInCondition (NW_FLAG_PERCIEVE_EVENT);' and uncomment it (remove the '/' double slashes at the beginning of the line). Compile the new script.

2) In your NPC's dialogue file, go to the very first line where he starts his dialogue with the PC. If you only want him to start this with any given PC once, add the following script into 'Actions Taken':

NWScript:

```
void main()
{
    SetLocalInt(GetPCSpeaker(), "Dlg_Init_" + GetTag(OBJECT_SELF),
    TRUE);
}
```

This sets the variable that is used in the script you were using, so the dialogue isn't done more than once.

3) Now create a new script in the OnUserDefined event for your NPC. You can put that second script in here, as follows:

NWScript:

```
void main()
{
    int nEvent = GetUserDefinedEventNumber();
    if (nEvent == 1002) // OnPerceive event
    {
        object oPC = GetLastPerceived();
        if (GetIsPC(oPC) && GetLocalInt(oPC, "Dlg_Init_" +
        GetTag(OBJECT_SELF)) == FALSE && !IsInConversation(OBJECT_SELF))
        {
            ClearAllActions();
            AssignCommand(oPC, ClearAllActions());
            ActionMoveToObject(oPC);
        }
    }
}
```

```

        ActionStartConversation(oPC);
    }
}

```

POSSIBLE DRAWBACK: Now... one reason why this may not work? If your PC starts the game visible to this NPC, there is no OnPerceive event. The event only fires if the PC was previously non-perceived and then became perceived.

Using the Trigger Method for starting a Conversation

This is the one we use in the official campaign. It allows you to control at which point the NPC will run up to the PC, and also determine how far the NPC will run after the PC before giving up.

To set this up, you need to do the following:

1) Like as in #2 above, in your NPC's dialogue file, go to the very first line where he starts his dialogue with the PC. If you only want him to start this with any given PC once, add the script from #2 above into 'Actions Taken':

NWScript:

```

void main()
{
    SetLocalInt(GetPCSpeaker(), "Dlg_Init_" + GetTag(OBJECT_SELF),
    TRUE);
}

```

2) Go into your toolset. Put down a waypoint at the spot where your NPC will return to. Label it "WP_RETURN_" + the tag of the NPC. (so if the tag is 'Fred', label it "WP_RETURN_Fred". Remember that all tags are case-sensitive!

3) While in the toolset, create a new generic trigger and draw a polygon around your NPC. When the PC crosses into this area, the NPC will start running toward the PC to talk. If the NPC leaves this area, he will stop chasing the PC and return to the waypoint.

4) Go to the trigger's 'Scripts' tab and include the following script under 'OnEnter':

NWScript:

```

// this is the on enter script if a trigger that
// encompasses the NPC who will be initiating dialogue.
// Make sure to replace "NPC_TALKER" with the actual tag of the
NPC
void main()

```

```

{
    object oNPC = GetObjectByTag("NPC_TALKER");
    object oPC = GetEnteringObject();
    if(GetIsPC(oPC) &&
    GetLocalInt(oPC,"Dlg_Init_" + GetTag(oNPC)) == FALSE &&
    !IsInConversation(oNPC))
    {
        AssignCommand(oPC,ClearAllActions());
        AssignCommand(oNPC,ClearAllActions());
        AssignCommand(oNPC,ActionMoveToObject(oPC));
        AssignCommand(oNPC,ActionStartConversation(oPC));
    }
}

```

5) Now you want to prevent the NPC from chasing after the PC. Enter the following under the trigger's OnExit event:

NWScript:

```

// This will return the NPC to a starting position
// if he attempts to leave the trigger.
// You must replace "NPC_TALKER" with the tag of the NPC.
// You must also have a waypoint with the tag "WP_RETURN_" +
NPC's Tag.
// This should be placed in the spot the NPC starts at.
void main()
{
    string sTag = "NPC_TALKER";
    object oExit = GetExitingObject();
    if(GetTag(oExit) == sTag)
    {
        AssignCommand(oExit,ClearAllActions());

        AssignCommand(oExit,ActionMoveToObject(GetNearestObjectByTag("WP_
RETURN_" + sTag)));
    }
}

```

This method doesn't care if the PC is visible or not... crossing into the trigger will cause the NPC to initiate. If you want to make sure that the NPC sees the PC, add the following line immediately under 'if(GetIsPC(oPC) &&'' in the OnEnter script:

GetObjectSeen(oPC, oNPC) &&

As well, in a multiplayer game, the NPC will attempt to talk to each and every player at least one with this method. If you only want him to run up to one PC EVER, then do the following:

1) In the dialogue script, replace 'GetPCSpeaker()' with 'OBJECT_SELF'.

2) In the OnUserDefined script (in the first method) or the OnEnter script (in the second method), replace the 'oPC' in the GetLocalInt command to 'oNPC'.

HOW DO I MAKE MY NPC ATTACK THE PC HE IS TALKING TO?

You want to add a script that makes the NPC go hostile. Place the script in the 'Actions Taken' tab down on the bottom right of the dialogue editor... make sure you have the node selected where you want the battle to begin.

If you want the NPC's faction to become hostile against the target, use the generic script "nw_d1_attonend":

NWScript:

```
///:////////////////////////////////////
///: Attack on End of Conversation
///: NW_D1_AttOnEnd
///: Copyright (c) 2001 Bioware Corp.
///:////////////////////////////////////
/*
    This script makes an NPC attack the person
    they are currently talking with.
*/
///:////////////////////////////////////
///: Created By: Preston Watamaniuk
///: Created On: Nov 7, 2001
///:////////////////////////////////////
#include "NW_IO_GENERIC"
void main()
{
    AdjustReputation(GetPCSpeaker(), OBJECT_SELF, -100);
    DetermineCombatRound();
}
```

If, however, you just want the sole NPC to become hostile and attack, use the generic script "nw_d1_attonend02":

NWScript:

```
///:////////////////////////////////////
///: Attack on End of Conversation
///: NW_D1_AttOnEnd02
///: Copyright (c) 2001 Bioware Corp.
///:////////////////////////////////////
/*
    This script makes an NPC attack the person
    they are currently talking with. This will
    only make the single character hostile not
    their entire faction.
*/
///:////////////////////////////////////
///: Created By: Preston Watamaniuk
///: Created On: Nov 7, 2001
///:////////////////////////////////////
```

```
#include "NW_I0_GENERIC"  
void main()  
{  
    SetIsTemporaryEnemy(GetPCSpeaker());  
    DetermineCombatRound();  
}
```

One thing to beware of: if you apply this script to the actual node where the NPC is speaking, they will go hostile right as the dialogue of that node is put up. Likely the NPC will not have time to read it all if it is a long piece of dialogue.

If you are OK with that, great. If not, use the Add button on the last NPC node to add a PC response. Erase the text in the PC response and press Enter... it should become an 'End Dialogue' node. Add the script to the 'Actions Taken' area for the End Dialogue, instead. This way, the player can read the dialogue the NPC is speaking... and when he presses 'Enter to Continue', the NPC will attack.

COMMON PROBLEMS: Make sure that a creature you want to go into combat does *not* have levels in Commoner. Commoners are set up by the generic AI to run away from hostiles and only form into a mob when there is both a level 10+ Commoner nearby. Also make sure that if you are using the 'Special Behaviors' in the OnSpawn script (one of the special behavior command lines is uncommented) that you are not using the 'Herbivore' behavior. Herbivores also never engage in combat and will always run away.

HOW DO I START MY STORE?

Setting up the basic store is pretty easy... all you have to do is use the Store Wizard and that gets your basic store object set up: you've got a store with an inventory, a mark-up and mark-down percentage, a name and a tag. But what do you do with it, then?

Step 1: Place the store object in your module A store that is just an unplaced blueprint cannot be accessed by your module. You've actually got to put it somewhere. You'll notice that when you place it, it looks just like a waypoint... that's fine. Players won't see this store 'object' or be able to interact with it in any fashion... it just needs to exist, preferably close by to the object that is going to call on it.

Step 2: Create your accessing object Usually this is a merchant NPC... but it doesn't have to be, really. Any object that can run a script can open a store. Having a merchant NPC is the easiest method, however.

Simply create your merchant and go to create a dialogue for him. The simplest method is simply have him say "Greetings! How would you like to see my wares?" Then add two player responses: "Yes" and "No".

Step 3: Add the script to open the store object In the above dialogue, you'd want to add this script into the 'Actions Taken' section of the "Yes" player response... so when the player selects that response, the store is opened. Nothing needs to go in "No"... that will simply end the conversation.

The basic script for opening a store object off of dialogue is this:

NWScript:

```
void main()
{
    object oStore = GetObjectByTag("STORE_TAG_HERE");
    OpenStore(oStore, GetPCSpeaker());
}
```

...and that's it. Naturally you want to replace "STORE_TAG_HERE" with the specific tag of your store object.

If you look at the command structure of OpenStore, you will see that you can optionally add on extra mark-up or mark-down percentage... that's just in case you want conditional store usage. Take this next script, for instance... the storekeeper doubles all prices for elves (I guess he doesn't like them):

NWScript:

```
void main()
{
```

```
object oStore = GetObjectByTag("STORE_TAG_HERE");
if (GetRacialType(GetPCSpeaker()) == RACIAL_TYPE_ELF)
{
    OpenStore(oStore, GetPCSpeaker(), 100);
}
else OpenStore(oStore, GetPCSpeaker());
}
```

Last comment on stores: the features that the standard store object offers are pretty limited at the moment. Perhaps that will change in the future. Regardless, know that you cannot adjust prices on individual types of items... a merchant applies the same mark-up or mark-down regardless of the item involved... nor can you dictate what types of items a merchant will or will not buy. It is all or nothing.

One thing that *does* work on stores is that any scripting command that targets an object's inventory works on the store object, as well. Remember: it's the store object that has the inventory, NOT the merchant NPC. So using `CreateItemOnObject`, `GetFirstItemInInventory` and `GetNextItemInInventory` will all work when targeting a store object.

USING MODULE EVENTS

When events are triggered (whether on a module, a placeable object, or wherever), the most important part to using the event via script is being able to identify who or what triggered the event. Keep in mind that this is separate from the UserDefinedEvent... these are the events that are hard-coded into each individual object. You assign a script to that event and, when it is triggered, that script will run.

Module Event List

Event: **OnAcquireItem**, *trigger*: whenever an item is added to someone's inventory, *what triggered it?* Use `GetModuleItemAcquired()`. *Also used here:* `GetModuleItemAcquiredFrom()` to return the object the item was taken from, `GetItemPossessor(object oItem)` to return who gained the item.

Event: **OnActivateItem**, *trigger*: whenever an item using the item property spell 'Unique Power' (which is targeted) or 'Unique Power - Self Only' casts that spell, *what triggered it?* Use `GetItemActivated()`. *Also used here:* `GetItemActivatedTarget()` to determine the target of a 'Unique Power' spell if the target was an object, `GetItemActivatedTargetLocation()` to determine the location target of a 'Unique Power' spell if the target was a point, `GetItemActivator()` to determine who owns the item that was activated.

Event: **OnClientEnter**, *trigger*: whenever a player logs into the module, *what triggered it?* Use `GetEnteringObject()` to return the player object.

Event: **OnClientLeave**, *trigger*: whenever a player logs out of the module, *what triggered it?* Use `GetExitingObject()` to return the player object.

Event: **OnModuleLoad**, *trigger*: when the module is first started, *what triggered it?* The module itself... there is no specific object to 'get' in this case. In effect, this is the module version of the `OnSpawn` script, run only once when the module is loaded.

Event: **OnPlayerDeath**, *trigger*: a player character death, *what triggered it?* Use object `GetLastPlayerDied()` to return the player object who has died. *Default use:* the "nw_o0_death" script is generally used here, which clears the relationship of the standard factions to the player and determines when the 'death GUI panel' pops up that allows the player to exit or respawn.

Event: **OnPlayerDying**, *trigger*: a player has been reduced below 0 hit points, *what triggered it?* Use `GetLastPlayerDying()`. The difference between this and `OnPlayerDeath` is that here the PC is out of HP but hasn't officially 'died' yet. *Default use:* The standard script placed here is "nw_o0_dying" which applies `EffectDeath` to the player and kills him officially to trigger the `OnPlayerDeath` event. If this script was removed, a player would bleed to death before officially dying at -10 hit points.

Event: **OnPlayerLevelUp**, *trigger*: a player has leveled up, *what triggered it?* Use object

GetPCLevellingUp(). This returns a player who has leveled up... but there is no way to prevent the player from finishing the process, as doing so is instantaneous and not an action like resting is.

Event: **OnPlayerRespawn**, *trigger*: a player has pressed the 'respawn' option on their Death GUI panel, *what triggered it?* Use GetLastRespawnButtonPresser(). *Default use*: the script "nw_o0_respawn", which resurrects the player, removes all adverse effects on him and applies the XP and gold penalty. It also determines if there is someplace (like a waypoint) with the tag "NW_DEATH_TEMPLE" and sends them there... if not, it resurrects him where he died.

Event: **OnPlayerRest**, *trigger*: a player has pressed the rest button, *what triggered it?* Use object GetLastPCRested(). *Also used here*: int GetLastRestEventType() --- note that this event fires both before and after a PC rest. You can detect which through the use of GetLastRestEventType(), with the constants REST_EVENTTYPE_REST_CANCELLED, REST_EVENTTYPE_REST_FINISHED, REST_EVENTTYPE_REST_INVALID, REST_EVENTTYPE_REST_STARTED. If you are looking to control when a player can rest, you want to detect if the rest event has started... and if it is not the time to do so, use the AssignCommand on the PC to tell him to ClearAllActions. This will stop him from completing his rest.

Event: **OnUnAcquireItem**, *trigger*: an item has been dropped or removed, *what triggered it?* Use GetModuleItemLost() to return the item. *Also used here*: GetModuleItemLostBy() to return the object that used to have that item.

Event: **OnHeartbeat**, *trigger*: every six seconds after module load, the module's heartbeat will fire. There is nothing that triggers this event... whatever script is in this area will run on every heartbeat. For that reason, be careful that you do not add large scripts into this area that will be running constantly, as the overhead on processing power that they can take up can be large.

Event: **OnUserDefined**, *trigger*: a script has used the SignalEvent command and targeted the module (probably via GetModule). The event sent must have been set to be a UserDefined event. (example of use: SignalEvent (GetModule(), EventUserDefined(100))). In a script placed here, you would use GetUserDefinedEventNumber() to watch for the number being sent... and on that occurring, initiate the rest of your script.

USING AREA EVENTS

Keep in mind that when you create scripts for the area events (or, indeed, for the module events) that the default 'performer' of any action-related commands is the area, itself.

If you refer to `OBJECT_SELF`, you are referring to the area object. If you use an action-related command such as `ActionMoveToObject` without using `AssignCommand` to first send that command to a creature object, you are telling the area to move (which it, of course, cannot). Simply keep this in mind as you code, so you can differentiate between telling the area what to do and telling things within the area what to do.

Area Events Listing

Event: **OnEnter**, *triggered by*: something (not necessarily a player character) has entered the area, *what triggered it?* Use `GetEnteringObject()` to return the object that has entered the area.

An example: Let's say you wanted to set an object variable called "PC_Entered" that was on the area whenever a player character entered the area (with the object being equal to the PC)... and then send a UserDefined event to a particular creature in the area with the unique tag "HUNTER" (I could then have an `OnUserDefined` event for that creature to come looking for the PC). You would place the following script in the `OnEnter` event for that area -

NWScript:

```
void main()
{
    object oEntering = GetEnteringObject();
    object oMonster = GetObjectByTag("HUNTER");
    // if the entering object is a player character
    if (GetIsPC(oEntering))
    {
        // set the object variable on the area to equal the PC
        SetLocalObject(OBJECT_SELF, "PC_Entered", oEntering);
        // send a UserDefinedEvent number 100 to the monster
        SignalEvent(oMonster, EventUserDefined(100));
    }
}
```

Event: **OnExit**, *triggered by*: someone has left the current area, *what triggered it?* Use `GetExitingObject()`.

An example: Let's take the same script above and set the "PC_Entered" area-set variable to an invalid object if the PC leaves. Let's also check if the exiting object is the monster from above (chasing the PC, no doubt)... and if it is, tell him to return to his cave (where there is a waypoint laid down with the tag "HUNTER_CAVE") -

NWScript:

```
void main()
{
    object oExiting = GetExitingObject();
    object oMonster = GetObjectByTag("HUNTER");
    object oCave = GetObjectByTag("HUNTER_CAVE");
    // if the exiting object is a player character
    if (GetIsPC(oExiting))
    {
        // set the area-set variable to an invalid object
        SetLocalObject(OBJECT_SELF, "PC_Entered", OBJECT_INVALID);
    }
    // if the exiting object is the monster
    if (oExiting == oMonster)
    {
        // tell him to stop what he's currently doing
        AssignCommand(oMonster, ClearAllActions());
        // tell him to move back to his cave waypoint
        AssignCommand(oMonster, ActionMoveToObject(oCave));
    }
}
```

Event: **OnHeartbeat**, *triggered by*: Nothing is required to trigger the heartbeat... any script that is placed here will be automatically triggered once every 6 seconds. As with the module events, be careful of placing scripts in here that would run constantly.

Event: **OnUserDefined**, *triggered by*: This can only be triggered by someone using the `SendSignal` command to specifically send a UserDefined event to the area object (usually via the `GetArea` command or `GetObjectByTag` using the area's tag).

USING CREATURE EVENTS

As mentioned previously in the 'UserDefinedEvents Are Your Friend' post, unless you know *exactly* what you are doing, it is not advisable to mess with the default AI scripts that are used on creatures (except for the OnSpawn script, which is what it's there for). It is perfectly possible to make use of every event listed below by uncommenting the associated flag in the OnSpawn script and writing an OnUserDefined script to make use of the associated event number (all the examples below will be OnUserDefined scripts doing so).

Creature Event Listing

Event: **OnPerception**, *triggered by*: something has entered the creature's perception radius. This does not automatically mean that the creature sees or hears this thing... just that it is possible for him to perceive it. *what triggered it?* Use GetLastPerceived() to return the last perceived creature (whether it was actually seen or not).

Also used here: GetLastPerceptionHeard() returns TRUE or FALSE as to whether the last perceived object was also heard, GetLastPerceptionInaudible() returns TRUE or FALSE as to whether the last perceived object has become inaudible, GetLastPerceptionSeen() returns TRUE or FALSE as to whether or not the last perceived object can be seen, GetLastPerceptionVanished() returns TRUE or FALSE as to whether or not the last perceived object can no longer be seen. One thing to remember: all the above commands relate **ONLY** to the OnPerception event. Do not use them elsewhere.

What does the default AI script do? The OnPerception generic AI gives priority to first going into search mode if an enemy has suddenly vanished, then telling the creature to combat an enemy that has appeared.

Example: I want my human NPC to turn towards anyone he sees, and if they are female to bow to them.

NWScript:

```
void main()
{
    int nEvent = GetUserDefinedEventNumber();
    // if the OnPerception event was fired
    if (nEvent == 1004) // OnPerception event
    {
        object oTarget = GetLastPerceived();
        // and I can see the person in perception range
        if (GetLastPerceptionSeen())
        {
            // put the command to turn towards them in my queue
            ActionDoCommand(SetFacingPoint(GetPosition(oTarget)));
        }
        // and if they are female
        if (GetGender(oTarget) == GENDER_FEMALE)
        {
            // pause for half a second
            ActionWait(0.5);
            // and then bow
        }
    }
}
```

```

        ActionPlayAnimation(ANIMATION_FIREFORGET_BOW);
    }
}
}

```

Event: **OnSpellCastAt**, *triggered by*: a spell has been cast at the creature... note that spells don't intrinsically cause this event. You will see in your command list a 'EventSpellCastAt'. The spell scripts specifically use this command on their target to tell it a spell has been cast... and the command also tells them whether or not the spell is a hostile one. *what triggered it?* Use `GetLastSpell()` to return the constant of the spell used (SPELL_*).

Also used here: `GetLastSpellCaster()` to return the object that cast the spell, `GetLastSpellHarmful()` returns TRUE or FALSE as to whether or not the event was marked hostile.

What does the default AI script do? If a harmful spell has been cast on the creature and it is not currently in combat, it goes hostile.

Event: **OnPhysicalAttacked**, *triggered by*: the creature has been attacked in melee, *what triggered it?* Use `GetLastAttacker()`. *Also used here(i):* `GetLastDamager()`
(i)*What does the default AI script do?* If the creature is attacked and not currently in combat, it goes hostile and emits the appropriate shouts to alert allies nearby.

Event: **OnDamaged**, *triggered by*: creature has lost hit points, *what triggered it?* Use `GetLastDamager()`.

Also used here[i/]: `GetTotalDamageDealt()`, `GetDamageDealtByType(int nDamageType)`, `GetCurrentHitPoints(object oObject = OBJECT_SELF)`, `GetMaxHitPoints(object oObject = OBJECT_SELF)`

(i)*What does the default AI script do?* if the creature was hurt by someone he can't see, try to find them. Otherwise locate an appropriate target and go hostile.

Example: with this particular creature, if he loses more than half his hit points he will shout out something, run to a waypoint called "CAVE_EXIT" and disappear.

NWScript:

```

void main()
{
    int nEvent = GetUserDefinedEventNumber();
    if (nEvent == 1006) // OnDamaged event
    {
        int nMaxHP = GetMaxHitPoints();
        int nCurHP = GetCurrentHitPoints();
        // if at less than half hit points
        if (nCurHP < (nMaxHP / 2))
        {
            // stop what I'm doing
            ClearAllActions();
            // cry out loud
            ActionSpeakString("Ahhh! Run!!");
        }
    }
}

```



```

        // run to the exit
        ActionMoveToObject(GetObjectByTag("CAVE_EXIT"), TRUE);
        // and destroy myself
        ActionDoCommand(DestroyObject(OBJECT_SELF));
        // accept no further AI commands from this point
        SetCommandable(FALSE);
    }
}
}

```

Event: **OnDisturbed**, *trigger*: something has either been added to or removed from the creature's inventory, *what triggered it?* Use `GetLastDisturbed()` to return the creature object who disturbed the inventory.

Also used here: `GetInventoryDisturbType()` will return either `INVENTORY_DISTURB_TYPE_ADDED`, `INVENTORY_DISTURB_TYPE_REMOVED`, or `INVENTORY_DISTURB_TYPE_STOLEN`, `GetInventoryDisturbItem()` will return the object that was either removed or added to inventory.

What does the default AI script do? Since a creature can't have an item added or removed from its inventory (it's not a container), if its inventory is disturbed it must have been stolen... go hostile if there is a valid target.

Event: **OnCombatRoundEnd**, *triggered by*: a combat round has ended and this event is triggered automatically. *What does the default AI script do?* Unless there is a special behavior set or the set warnings flag in `OnSpawn` has been set, a new combat round is initiated.

Event: **OnConversation**, *triggered by*: the creature has been clicked on for conversation OR someone has audibly spoken, *what triggered it?* Use `GetLastSpeaker()` to return the creature who clicked on them or spoke out loud.

Also used here: `GetListenPatternNumber()` to return a pattern number if one has been set up.

What does the default AI script do? If the creature has been clicked on for conversation, it will stop what it's doing and begin dialogue (if it can). Otherwise the script checks to see if a spoken phrase is recognizable and if the creature has been set up to recognize combat shouts.

Event: **OnRested**, *triggered by*: the creature has rested via `ActionRest()`. Creatures in the game don't normally rest, so there is no default AI script functions here.

Event: **OnDeath**, *triggered by*: the creature has been killed, *what triggered it?* use `GetLastKiller()` to return the object that killed the creature. Note that any scripts that are run off this event need to be immediate... using `DelayCommand` will not work as the creature cannot run commands past the point of its death.

What does the default AI script do? It checks to see if the creature was non-evil and had Commoner levels... and if so automatically changes the killer's alignment a little towards evil. It also emits standard shouts for help to any creatures within range who are set up to

listen for them.

Event: **OnBlocked**, *triggered by*: a door is blocking the creature's movement, *what triggered it?* Use `GetBlockingDoor()` to return the door object. Note that only doors currently trigger the `OnBlocked` event... this is not used for collision detection (yet). *What does the default AI script do?* If the creature is smart enough to use a door (Intelligence of 5+) it will first attempt to open the door and, if that's not possible, will attempt to bash it down.

Event: **OnHeartbeat**, *triggered by*: there is no trigger needed for the heartbeat... any script in this area will be run automatically once every six seconds. It is important that you do not attach scripts to the heartbeat which will run constantly, as the resulting overhead (especially if you have a lot of creatures doing so) can be high. *What does the default AI script do?* It checks for some conditions that are set in the `OnSpawn` and executes them if they are applicable... such as the 'fast buff' for enemies (used when PC's draw within a certain range), day/night postings for the `WalkWayPoints` function and the standard ambient animations used by creatures when standing around.

USING PLACEABLE OBJECT EVENTS

Like with areas, the thing to remember when writing scripts for placeable objects is that when you using 'action'-type commands (commands that return void) the default performer is the placeable object, itself. If you want the script to tell something other than the placeable object to perform a command, you need to use the AssignCommand action. Also keep in mind that while placeable objects are capable of performing some actions (such as ActionPlayAnimation with reference to ANIMATION_PLACEABLE_*... although not all placeables open/close or activate/deactivate), telling them to do things that are inherently impossible often causes problems (ActionLockObject, ActionUnlockObject, ActionEquipItem, ActionPickUpItem... these are examples of things you should never tell a placeable object to do).

Placeable Object Event Listing

Event: **OnClose**, *trigger*: the placeable has been close (not that only placeables that can be opened or closed will fire this... like a chest or armoire), *what triggered it?* use GetLastClosedBy() to return the object that closed the placeable.

Example: here is an OnClose script that checks to see if its inventory is empty after it's closed... and, if so, creates 100 gold inside.

NWScript:

```
void main()
{
    // get the first item in inventory
    object oInside = GetFirstItemInInventory();
    // and if there isn't one
    if (!GetIsObjectValid(oInside))
    {
        // create the gold piece item with a stack of 100
        CreateItemOnObject("NW_IT_GOLD001", OBJECT_SELF, 100);
    }
}
```

Event: **OnDamaged**, *trigger*: the placeable object has taken damage, *what triggered it?* Use GetLastDamager() to return the object that damaged the placeable. *Also used here*: GetTotalDamageDealt(), GetDamageDealtByType (int nDamageType), GetCurrentHitPoints(), GetMaxHitPoints().

Event: **OnDeath**, *trigger*: the object has been destroyed (playing its 'destruction' animation is automatic), *what triggered it?* GetLastKiller() will return the object that destroyed the placeable.

Event: **OnHeartbeat**, *trigger*: there is no trigger for the heartbeat... any script in this location fires automatically once every six seconds. Be careful not to put scripts in this

area that will run constantly, as the overhead can be high.

Event: **OnDisturbed**, *trigger*: an item has either been added to or removed from the placeable's inventory, *what triggered it?* Use `GetInventoryDisturbType()`. This results in one of these constants: `INVENTORY_DISTURB_TYPE_ADDED`, `INVENTORY_DISTURB_TYPE_REMOVED`, or `INVENTORY_DISTURB_TYPE_STOLEN`. *Also used here*: `GetLastDisturbed()` returns the object that removed/added the item, `GetInventoryDisturbItem()` returns the item that was added/removed.

example: here's an `OnDisturbed` script that sends a `UserDefinedEvent #500` signal to a wizard (with the tag "WIZARD1") if anyone removes a particular potion (with the tag "PRECIOUS_POTION") from the placeable.

NWScript:

```
void main()
{
    object oWizard = GetObjectByTag("WIZARD1");
    object oPC = GetLastDisturbed();
    object oPotion = GetObjectByTag("PRECIOUS_POTION");
    // if the removing object is valid and the potion is removed
    if (GetIsObjectValid(oPC) &&
        (GetInventoryDisturbType() ==
INVENTORY_DISTURB_TYPE_REMOVED) &&
        (GetInventoryDisturbItem() == oPotion))
    {
        // send the signal to the wizard
        SignalEvent(oWizard, EventUserDefined(500));
    }
}
```

Event: **OnLock**, *trigger*: the placeable has been locked (or relocked), *what triggered it?* Use `GetLastLocked()` to return the object that locked the placeable. *Also Used Here*: `GetLockLockDC()` returns the DC for locking the placeable. ****NOTE**** I've had a few reports that this event may not be working.

Event: **OnPhysicalAttacked**, *trigger*: the placeable has been physically attacked (but not necessarily damaged), *what triggered it?* Use `GetLastAttacker()`.

Event: **OnOpen**, *trigger*: the placeable object has been opened (for those placeable objects that can be opened and have their 'Has Inventory' box checked), *what triggered it?* Use `GetLastOpenedBy()`.

Event: **OnSpellCastAt**, *trigger*: a spell has been cast at the placeable, *what triggered it?* Use `GetLastSpellCaster()`. *Also used here*: `GetLastSpell()` will return the constant of the spell used (`SPELL_*`), `GetLastSpellHarmful()` will return `TRUE` or `FALSE` depending on whether or not the last spell cast was marked hostile.

Event: **OnUnlock**, *trigger*: the placeable has been unlocked, *what triggered it?* Use `GetLastUnlocked()` to return the object that unlocked the placeable. ****NOTE**** I've had a few reports that this event may not be working.

Event: **OnUsed**, *trigger*: if a placeable object has its 'Useable' box checked, then clicking on the object sets off this event, *what triggered it?* Use `GetLastUsedBy()`.

example: a script for a placeable object that heals whoever used it.

NWScript:

```
void main()
{
    object oUser = GetLastUsedBy();
    effect eHeal = EffectHeal(GetMaxHitPoints(oUser));
    effect eVis = EffectVisualEffect(VFX_IMP_PULSE_HOLY);
    // if the using object is valid
    if (GetIsObjectValid(oUser))
    {
        // apply the visual effect
        ApplyEffectToObject(DURATION_TYPE_INSTANT, eVis, oUser);
        // and, 1 second later, heal the user
        DelayCommand(1.0, ApplyEffectToObject(DURATION_TYPE_INSTANT,
        eHeal, oUser));
    }
}
```

Event: **OnUserDefined**, *trigger*: this event will only occur if someone specifically uses `SignalEvent` to send a `UserDefinedEvent` to the placeable object.

USING TRIGGER EVENTS

Again, make sure that any actions you wish to be caused by the trigger are sent to the appropriate targets via AssignCommand... or else you are telling the trigger, itself, to do something, and there is very little that a trigger can do on its own.

Trigger Event Listing

Event: **OnClick**, *trigger*: if you want a trigger that will cause an effect when it is clicked upon by the player, a script must be entered in this area. Most types of area transitions use this event (although a trigger that is set up as the 'Area Transition' type can make use of the wizard to do so). *who triggered it?* Use GetClickingObject() to return the player object who clicked on the trigger.

Event: **OnEnter**, *trigger*: any object that crosses the 'border' of the trigger will set off this event. *who triggered it?* Use GetEnteringObject().

Example: the following OnEnter trigger script will jump every member of the party who is in the same area to the waypoint tagged "JUMP_HERE".

NWScript:

```
void main()
{
    object oWP = GetObjectByTag("JUMP_HERE");
    object oParty = GetFirstFactionMember(GetEnteringObject(),
TRUE);
    // cycle through valid player character party members
    while (GetIsObjectValid(oParty))
    {
        // and if they are in the same area
        if (GetArea(oParty) == GetArea(GetEnteringObject()))
        {
            // jump the party member to the waypoint
            AssignCommand(oParty, JumpToObject(oWP));
        }
        // get the next party member
        oParty = GetNextFactionMember(GetEnteringObject(), TRUE);
    }
    oParty = GetFirstFactionMember(GetEnteringObject(), FALSE);
    // now cycle through all non-player party members (like
familiar)
    while (GetIsObjectValid(oParty))
    {
        // and if they are in the same area
        if (GetArea(oParty) == GetArea(GetEnteringObject()))
        {
            // jump the party member to the waypoint
            AssignCommand(oParty, JumpToObject(oWP));
        }
        // get the next party member
        oParty = GetNextFactionMember(GetEnteringObject(), FALSE);
    }
}
```

```

// now jump the entering player to the waypoint
// this line may not be necessary... including it
// just to make certain the original PC jumps
AssignCommand(GetEnteringObject(), JumpToObject(oWP));
}

```

Event: **OnExit**, *trigger*: whenever an object crosses the boundary of the trigger to leave the trigger, *who triggered it?* Use `GetExitingObject()` to return the exiting creature.

Example: this `OnExit` trigger script sends a command to any exiting giant to return to a waypoint.

NWScript:

```

void main()
{
    object oWP = GetObjectByTag("RETURN_HERE");
    object oCreature = GetExitingObject();
    // if the exiting creature is a giant
    if (GetRacialType(oCreature) == RACIAL_TYPE_GIANT)
    {
        // tell it to stop what it's doing
        AssignCommand(oCreature, ClearAllActions());
        // and move to the waypoint
        AssignCommand(oCreature, ActionMoveToObject(oWP));
    }
}

```

Event: **OnHeartbeat**, *trigger*: like other heartbeats, the heartbeat of a trigger activates every 6 seconds. A trigger, however, is not quite the same as another object... it is only 'active' when something is within its bounds or it has been clicked upon. You cannot place a trigger and give it a heartbeat script to constantly issue orders... if you want something to do that, use an invisible placeable object, instead.

Event: **OnUserDefined**, *trigger*: again, this event will only fire if a `UserDefinedEvent` has been specifically sent to the trigger via the command `SignalEvent`.

[Edited By David Gaider: Monday, 08 July 07:18PM (GMT)]

QUEST CREATION WALKTHROUGH - NO SCRIPTING REQUIRED

by Iskander

There have been a bunch of request along the lines of:

Quote:

Help! I want to make a quest, but I don't know how. I want to have:

- 1) NPC says they need you to get something for them
- 2) PC gets item and brings it back
- 3) NPC rewards PC with gold or experience, alters dialogue to reflect new state

Here's one way to do this, using only the script wizards and the conversation editor. **You don't need to edit a single line of script for this.**

You're going to use a local variable on the PC to determine the state of the quest (for him/her); we'll call it "iPlottQuest". It doesn't matter what it's called (as long as it doesn't conflict with an NWN default value). iPlottsQuest is going to have 3 possible values:

Quote:

- 0** - The PC hasn't been given the quest (default).
- 1** - The PC has been given the quest, but hasn't completed it.
- 2** - The PC has finished the quest, and can't do it again.

- Paint your NPC, let's call him Plott deVice
- Paint your Item elsewhere, and edit its properties. Give it a unique tag, which we'll be using later. Let's call it 'itemPlottsDoohickey'.
- Create a new conversation for Plott deVice
- Add the following lines of dialogue to the conversation root *in this order*:

Quote:

- Thank you again, you're wonderful.[END DIALOGUE]
- Have you found the Doohickey of Destruction?[END DIALOGUE]
- Excuse me, could you help me out?[END DIALOGUE]

- Note that the lines of dialogue are backwards because when faced with multiple choices for an NPC at the same conversation level, the engine will choose the first line of dialogue that either has no 'Text Appears When...' script, or whose 'Text Appears When...' script evaluates to TRUE. Also, you don't need to put the 'END DIALOGUE' bits in, the editor will do that automatically.

Now we just need to set the conditions under which each line of dialogue appears. The last one is easy - it's a default. If neither of the first two lines appears, the last one always will, so we're not going to change that.

- Select the 'Thank you' line, and look at its 'Text Appears When...' tab. Click on the wizard's hat button to start the Script Wizard. Since we're checking to see if the quest is done, we're going to look for the local variable we chose, so check 'Local Variable' and click Next.
- In the '**What local variables have to be set?**' screen, we need to fill in some blanks. When we're done, the five boxes at the top should read like a (strange) sentence that answers the question. In this case, fill in the blanks so it reads "int *iPlottQuest* is equal to constant int 2", then click 'Add'.
- The wizard will add a line to the 'Local Expressions' box that should read **GetLocalInt(GetPCSpeaker(), "iPlottQuest") == 1**. Click 'Next'.
- Give your script a name. At this point, it's not particularly important what the name is, especially if you're only using the wizards. Click Finish.
- Now the first line of dialogue will only show up when the PC has completed the quest.
- Repeat the process with the 'Have you found...' line, but set the value to **1** instead of **2**. This line will only appear when the PC has been given the quest, but hasn't got the Doohickey yet.
- Now you need to flesh out the conversation to give the PC paths of response. PC's got to be able to accept the quest, and say 'Yes' when he's got the item. Add lines of dialogue to the conversation so it looks like this:

NWScript:

```
--- Thank you again, you're wonderful.[END DIALOGUE]
++ Have you found the Doohickey of Destruction?
| |
| | -+ Yes.
| | |- Thank you, thank you, thank you. Would you accept this
small token of my esteem?[END DIALOGUE]
| | |- No you haven't! Please keep looking, I just know it's out
there.[END DIALOGUE]
| |
| | -- Not yet, still looking.[END DIALOGUE]
|
++ Excuse me, could you help me out?
|
| -+ Sure, what's the problem?
| |
| | -+ I've lost my Doohickey of Destruction, can you find it
for me?
| |
| | - Yep. Be right back.[END DIALOGUE]
| | - Sorry, too busy... gotta go![END DIALOGUE]
|
| -- Sorry, too busy... gotta go![END DIALOGUE]
```

- Now, let's start the PC's quest. Highlight the 'Yep. Be right back' line, and go to its '**Actions Taken**' tab. This is where you specify what should happen when a PC (or NPC) says a particular line. Once again, click on the hat to start the Script Wizard.
- Since we're modifying the flag on the PC to let us know that the quest has started, check the 'Set Local Variables' box, and click 'Next'.
- Fill in the blanks so the boxes read "int *iPlottQuest* constant int **1**", and click 'Add', 'Next' and save the script. Now, when the PC chooses 'Yep. Be right back' in conversation, the engine will set that PC's 'iPlottQuest' variable to **1**.
- Finally, we need to complete the quest, and reward the PC. First, we add a conditional on the 'Thank you, thank you' line to check whether the PC has really got the item. Go to the 'Text Appears When...' tab, and start the wizard, but this time check the 'Item in Inventory' box, and click 'Next'.
- Fortunately, you jotted down the tag of your special item, and here's where you use it: type 'itemPlottsDoohickey' in the 'Enter a new tag' box, and click 'Add'. Save the script. Note that you don't need to add a conditional for the "No you haven't!" line, because it's the default.
- Finally, we get to the rewards. Still on the 'Thank you, thank you' line, switch to the 'Actions Taken' tab, and start the Wizard. We've got quite a lot to do here, so check the 'Give Rewards', 'Take from Player', and 'Set Local Variable' boxes, before clicking 'Next'.
- In the '**Give what rewards?**' screen... give rewards, and decide whether the party or just the PC gets the benefits. Click 'Next'.
- In the '**Take what?!**' screen, type 'iPlottsDoohickey' into the 'Take item (by Tag)' box, and leave 'Destroy' selected. Click 'Next'.
- Finally, we need to set our flag variable for the last time. Set 'iPlottsQuest' to **2**, click 'Add', 'Next', save the script, and we're done.
- Load the module into single-player, and go chat to your NPC.

This is an elementary Fetch! quest, without a single line of script-writing necessary. You do need to understand a little bit about variables, and tags, and you need to be aware that the Conversation Editor and the Script Wizard is writing a bunch of little scripts behind the scenes (you'll see they've been added to your module already). But you **don't need to write the scripts yourself**. Which is cool.

Here are some variations, all achievable with the wizards, that you might like to try:

- You've given the PC money and experience... how about improving his reputation with the NPC's faction?
- Give the PC the chance to insult the NPC, and have that change the PC's reputation with the NPC's faction for the worse.

- Add a '<StartHighlight>[Keep the Doohickey]</Start>Nope... couldn't find it, honest!' line of dialogue to the 'Have you found...' tree. Make it appear only when the PC has the Doohickey.. and add some consequences to the theft.
- Make the NPC xenophobic: only start the quest if the PC is the same race.
- Add a Persuade option for PCs to try and get the quest from the xenophobic NPC.
- Make it so that the Persuade check doesn't appear for PCs with low Charisma.
- Add another NPC to the quest, one that knows where the Doohickey is. (You'll need to add some more states to iPlottQuest and have that NPC modify the same Local PC variable in its conversation).

Finally, some tips:

(list)

(*) To get conversations to loop back to an earlier point, you can use linking. Copy the node you want to link to, and then right-click the node you want to link from... you'll see a 'Paste as link' option. *(Note that you can't paste links from a speaker onto its own dialogue)*

(*) To break long NPC descriptions up, add a single child conversation node, delete the default text, and leave it blank. Then continue the NPC's waffle in a child of the blank node. You'll see the node change to '[CONTINUE]' in the editor... the PC gets the familiar 'Continue.' option in the conversation.

(/list)

I hope this is of use to someone... if you do use it, and find anything unclear, excessively patronizing, or otherwise objectionable, please let me know, and I'll try to fix it.

Cheers,
Iskander

[Edited By David Gaider: Tuesday, 09 July 03:12AM (GMT)]

Celowin's Scripting Tutorial Lesson III - Conditionals

Introduction

The purpose of this sequence of lessons is to take a complete beginner to programming, and teach him or her how to use NWScript to write modules. The early lessons will be very basic, and anyone that has done any coding at all will be able to skip over them. The goal here is to make the lessons so that even the people that just shudder at any type of code can learn.

Feel free to post these lessons on any forum, print them out, or modify them. However, just give me credit for doing them.

Any comments on these lessons, good or bad, can be sent to me at james.foxglove@verizon.net

I am going to assume that anyone looking at these lessons has at least played around with the Aurora Toolset a bit. If there is enough feedback that people don't know how to do the simple placements that I have in these lessons, I will consider spelling out in more detail what needs to be done.

Let's Begin

This lesson is going to cover what is probably the most important part of scripting... how to make your code do something only some of the time – that is, only when some *condition* is met. At a guess, on looking through the forum, 90% or so of the questions come down in one way or another to dealing with this.

I've already previewed this idea at the end of Lesson 2, but I didn't give any explanation. The script I really want to showcase in this lesson is fairly complicated, so I'm going to go ahead and talk a bit of theory first.

The basic format for a conditional, or "if statement" is like this:

```
NWScript:
    if ( condition )
    {
        do_this;
        do_something_else;
        do_another_thing;
    }
```

(The above is not a true script, just something to give the general idea.)

Basically, if the "condition" is met, the script will do the things between the { and the }. If the condition isn't true, it does *nothing*.

(Note on the format: It is a common beginner's error to put a semicolon after the line starting with "if". There isn't supposed to be one there.)

That is really all there is to it... the tricky part is figuring out exactly what a "condition" is. There are many, many types of conditions we can put in here, with lots of little rules affecting how they work. I am certain that this lesson will not be able to cover every single rule that can be put here. I doubt that I myself know all of the functions that we could call. I can only hope to get you to the point where you know enough to understand the basics, so that you can look at other people's scripts and learn from them.

Most conditions come down to a comparison. For example, in the script I put at the end of Lesson 2, the relevant line was

```
if (nCount == 1)
```

Notice the "==" is in fact two equals signs. As we discussed in lesson 2, a single equals sign is used to "set" a variable. The double equals sign is more asking the question "are the two sides the same?"

Just looking at the statement, the sides look very different. One side has a bunch of text, the other side is a number. How can there be any question that they are the same?

Well, remember that nCount is a *variable* and is actually equal to a number. So, it all depends on the value stored in nCount. If nCount is storing the number 1, then the condition will be true, and the associated actions are performed. If nCount is storing the number 7, then the condition will be false, and this statement will not do anything.

The other major way to test a condition is through a predefined function. We'll explore that in this next example.

Example Script

Let's build a script with a simple conditional. There are going to be a number of new functions introduced in it, I'll explain them afterward.

We're going to need to do some setup for this script to work.

- Open up the Test Module we've been working with in the editor.
- Create a new area, forest tileset, dimensions 4 by 2. Call it Test Area 002.
- At one end, paint the module start location.
- At the other end, paint an npc. Just make it a commoner, for simplicity.
- Change the tag of the npc to GUARD
- Go to the scripts tab of the npc, delete all the scripts. (Next lesson, I hope, we can get rid of this step. It irks me to have to use npcs that don't react.)
- Go to the "OnPerceived" handle, and input the following script.

NWScript:

```
object oSeen = GetLastPerceived();
void main()
{
    if (GetIsPC(oSeen))
    {
        ActionSpeakString("Greetings, friend.");
    }
}
```

- Save it, using our standard naming convention, tm_guard_op (op for On Perceived)
- Ok everything on the npc, save the module, and go test it.

Run toward the guard.... when you get near, it will speak its phrase. Run far enough away and come back, it will speak it again. If you stay close to the npc, though, it won't do anything.

Let me switch into question and answer mode to try and explain this one. I'm using the same naming conventions used before, so I won't go into those. It is just the script that I'll focus on.

Another new handle, eh? What does this OnPerceived thing do?

This one calls the attached script whenever the npc notices something in game. If something is invisible or hiding, and the npc doesn't notice it, the script won't be called.

We want the npc to react when she sees a character, hence the use of this handle.

What is this object stuff in the first line? I didn't understand it the last time you threw something before main, and now you're pulling it again!

Well, really, we're using it the same way we did before, it is just a new "data type." Before, we set up a variable to store an integer. Now, we are setting up a variable to store an "object."

I've said it before, and I'll probably repeat it again later. Nearly everything in the game is an object. NPCs, players, items, waypoints, placeables... these are all objects. Many, many functions in the game are written just to deal with figuring out what object is what, and many more are written to manipulate objects.

So, we are setting up a temporary variable, which we are calling oSeen (the starting o to remind us it is an object), and storing a value into it.

What about the GetLastPerceived() part of the line?

This is a BioWare written function. Any of their functions that start with Get will return some sort of data. The names are usually rather descriptive... this function gives as an output the last object that was seen by the npc.

(As a note... I have a tough time thinking of any time you would use this function outside of the OnPerceived script handle. It gets used in just about every OnPerceived script, but basically never outside of it.)

So, putting this together with the last question, this first line of our script is just making it so that we can refer to the object that the npc saw in the first place.

You spent all that time up above talking about comparisons, and now you've only got one thing inside your if condition! What gives?

Well, this is a peculiarity of the GetIsPC function. It takes an object as an input, and returns TRUE if it is a PC, or FALSE if it isn't.

But this is exactly what we need for a condition! If it *is* a PC, the attached lines run. If it *isn't* a PC, then they won't.

If you prefer, you can change the line to

```
if (GetIsPC(oSeen)==TRUE)
```

to make it look more like a comparison... but as we've seen, it isn't really necessary.

Do we even need the if check at all? Won't it always be a PC that the npc notices?

For our little test module, yes. There is really nothing else in the module for the guard to perceive.

Everything I write in these lessons, though, I try to write in the same manner that I would for a real module. What if there was a hostile goblin nearby? You wouldn't want the guard to call it a friend.

For that matter, even friendly npcs...why bother talking to them if there is no PC around to see the interaction?

Aha! Now I've caught you! It won't be that all PCs are friendly either!

Good point. Let's play around with our module a bit.

The Changing of the Guard

(Sorry, couldn't resist the pun....)

What we're going to do is modify our module so that the guard will attack any pc it sees

that doesn't have a special ring. Only if the pc carries the ring will the guard call out the friendly greeting.

- Open the toolset, load up the module, and go to Test Area 002
- First, create the ring. Go to "Paint Items", "Miscellaneous", "Jewelry", "Rings", "Copper Ring." Place it near the module start.
- Edit the properties of the copper ring, change the tag to PASSRING
- If you're feeling ambitious, you can edit the name of the ring, give it a description, whatever. For the purposes of the script, only the tag matters.
- Now, go to the guard and open up the OnPerceived script we had before. Change it so that it is like this:

NWScript:

```
// Friend or Foe Script: tm_guard_op
// This should be placed in the OnPerceived handle of a guard.
//
// The guard will check to see if a PC has a passring, and if
not, attack.
object oSeen = GetLastPerceived();
object oRing = GetItemPossessedBy(oSeen, "PASSRING");
void main()
{
    // If it isn't a PC that the guard sees, it won't do anything.
    if (GetIsPC(oSeen))
    {
        if (oRing == OBJECT_INVALID)
        {
            // If the PC doesn't have the ring, attack the PC.
            ActionSpeakString("Die, trespasser!");
            ActionAttack(oSeen);
        }
        else
        {
            // Otherwise the PC does have the ring. Be friendly.
            ActionPlayAnimation(ANIMATION_FIREFORGET_GREETING);
            ActionSpeakString("Greetings, friend.");
        }
    }
}
```

I'm starting to embellish my scripts a bit, throwing in more and more new commands. While it may be a bit confusing at first, you'll really notice that if you start understanding the basic structure, all these random commands start to fall into place.

Anyway, save it all, load it up as a module, and see what happens. First, get the ring on the ground, and approach the guard. It should be friendly. Next, run away, drop the ring on the ground, and approach again. It will attack you this time.

Breaking it Down

Argh! You've added another new initialization! I hate that!

All I can say is to trust me, the scripts look a **lot** worse without them. So, let's take a look at this new initialization line.

```
object oRing = GetItemPossessedBy(oSeen, "PASSRING");
```

Once again, we're setting up a temporary variable, which will hold an object. The `GetItemPossessedBy` function takes in two inputs... the first is the creature object that you want to check for the item, the second is the tag of the item you're checking for.

We've already defined `oSeen` as the person triggering the script, the one getting noticed by the npc. So, `oRing` is the ring carried by that person that has the tag `PASSRING`.

But what if the person *doesn't* have the ring? What happens then?

Well, the `GetItemPossessedBy` still runs. But since it can't find the object on the person, it comes back with `OBJECT_INVALID`. Basically, a fancy way of saying "No such thing."

What are all these lines starting with //? They look like English instead of script?

Anything in a line after a `//` is ignored by the script. These are called "comments." Any good scripter will put in comments to explain what is going on.

It is really for your own good. You may perfectly understand a script when you write it... but that doesn't mean you'll remember every detail a month from then when you have to modify it.

Also, it is polite to do it for the sake of anyone else that will look at your script. The more explanation you give on what you are trying to do, the easier someone else will understand what you have written.

Your main script is looking really confusing. You have four sets of { and }! How am I supposed to keep track of all of them?

It can be tough, I freely admit it. The more complicated the script, the more confusing these "nested" statements can be. One trick that helps a lot is to use indentation, like I have been.

Some people like putting in blank lines to further break the script into "blocks." Again, I've done it a bit up there. I don't think it helps much in this script, but it takes no effort, so I may as well.

Comments can help break up a script into blocks as well.

Even with all these things, it can still be confusing. I'm not sure what more I can say, other than "you'll get better at reading them with practice."

What is this "else"?

"else" is an addition to the if-statement. The format becomes something like:

```
if ( condition )
{
do_this;
and_this;
}
else
{
do_this_instead;
and_also_this;
}
```

Basically, if the "if" part of it doesn't "go off", it does the "else" instead.

Let's analyze our whole inside if-statement. It checks... is the temporary variable oRing equal to OBJECT_INVALID? (That is, did the PC **not** have the ring?) If so, attack.

On the other hand, if that wasn't true, then the PC **did** have the ring. So, we do the "else" part, and be friendly.

For many people tracing through the logic of things like this is the hardest part of scripting. If you are a visual person, making a flowchart helps a lot. I would put one here, but I'm not going to try to draw one using ascii.

What are the new actions you put in here?

I think the names are rather descriptive. ActionAttack attacks the thing you pass to it as an input. ActionPlayAnimation has the npc perform an animation – in this case, the one called ANIMATION_FIREFORGET_GREETING. Basically, it is just the way to tell the npc to wave.

One More Modification

This lesson is getting huge, but there is one more thing that I want to throw in. (Ok, I lied... there are 5,217 more things that I want to throw in, but I'm trying to make this digestible.)

What if we want the *opposite* behavior from our guard? We want to attack if the PC has the ring, and be friendly if not? For example, maybe the ring was stolen?

We could do move a bunch of blocks of text around... and if we did it right, it would

work. But we can actually achieve this result by changing 1 character in our script. Where it says

```
if (oRing == OBJECT_INVALID)
```

change it to

```
if (oRing != OBJECT_INVALID)
```

"!=" is another kind of comparison. It checks to see if the two sides are *not* equal to each other. So now, if the PC *does* have the ring, the guard will attack.

Conclusion

There are a lot of new ideas in here, but once you have them mastered you really have unlocked the true power of NWScript. In particular, combining conditionals with local variables opens up all sorts of possibilities.

As always, feel free to ask questions about anything you don't understand. I do my best to explain things in a way that makes sense to everyone, but these are complicated ideas if you haven't dealt with them before. Sometimes, something that seems obvious to me can be confusing to someone else.

Lesson Four should be easier to write... I'll probably have it out soon. For those curious, I'm going to discuss User Defined Events, and how to make an npc without throwing away all of BioWare's hard work in writing the default scripts.

[Edited By David Gaider: Tuesday, 09 July 03:12AM (GMT)]

ADDING & REMOVING EFFECTS

If you scroll down the list of commands in your script editor, you will see a long list of commands that begin with the word 'Effect'. These are the basis for building all spells, special abilities and visual effects in the game. There are two commands that allow you to use them: `ApplyEffectToObject` and `ApplyEffectAtLocation`. Both of them are very similar in their structure, except the former targets a specific object (creature, item, placeable object, etc) and the other targets a specific location (a point in space).

void ApplyEffectToObject (int nDurationType, effect eEffect, object oTarget, float fDuration=0.0f)

Duration Type: the 'nDurationType' is asking for a constant that begins with `DURATION_TYPE_*`, of which there are only three. `DURATION_TYPE_INSTANT` would apply to any effect that is both immediate and permanent and cannot be removed. `EffectDamage` would be one example, as well as any of the 'fire-and-forget' visual effects. If not instant, the constant must be either `DURATION_TYPE_PERMANENT` or `DURATION_TYPE_TEMPORARY`. Permanent is as it sounds... the effect lasts until removed or dispelled. A temporary duration means that the effect must have a duration specified in the command (and is the only time this is so).

Effect: This is the specific effect that you are going to use. It is common to define an effect variable first which contains all the pertinent information of the effect command, and then use that variable when applying... but it is not required. Each effect will vary in its requirements. Some have only a set application and are simply called, such as `EffectSleep` or `EffectCharmed`. Some require additional parameters, such as `EffectDamage` or `EffectPolymorph`. I won't go through every effect here, but it is important that you provide all the parameters required by an effect or it will be invalid.

Target: this will either be the object (in `ApplyEffectToObject`) or the location (in `ApplyEffectAtLocation`). It can be declared separately in a variable or defined in the command.

Duration: a duration (in seconds) is only required for effects of `DURATION_TYPE_TEMPORARY`. When specified, it must be as a float... meaning the number must have a decimal place. So put '3.0' for 3 seconds, or '10.5' for 10 and a half seconds.

Some Examples of Effects

- on the use of an object, polymorph the user into a pixie

```
NWScript:
    void main()
    {
```

```

        effect ePixie = EffectPolymorph(POLYMORPH_TYPE_PIXIE);
        ApplyEffectToObject(DURATION_TYPE_PERMANENT, ePixie,
        GetLastUsedBy());
    }

```

- an item with the Unique Spell property (with the tag "WONDERFUL_WAND" that heals and hastes the target for 30 seconds:

NWScript:

```

void main()
{
    object oWand = GetItemActivated();
    if (oWand == GetObjectByTag("WONDERFUL_WAND"))
    {
        object oTarget = GetItemActivatedTarget();
        int nMaxHP = GetMaxHitPoints(oTarget);
        int nCurHP = GetCurrentHitPoints(oTarget);
        effect eHeal = EffectHeal(nMaxHP - nCurHP);
        ApplyEffectToObject(DURATION_TYPE_INSTANT, eHeal,
        oTarget);
        ApplyEffectToObject(DURATION_TYPE_TEMPORARY,
        EffectHaste(), oTarget, 30.0);
    }
}

```

- in a creature's OnUserDefined, when an event #100 is sent, a generic bugbear is summoned without effects at a waypoint tagged "SUMMON_POINT":

NWScript:

```

void main()
{
    int nUser = GetUserDefinedEventNumber();
    if(nUser == 100)
    {
        effect eSummon = EffectSummonCreature("bugbeara001");
        location lWP =
        GetLocation(GetObjectByTag("SUMMON_POINT"));
        ApplyEffectAtLocation(DURATION_TYPE_PERMANENT, eSummon,
        lWP);
    }
}

```

Visual Effects

Visual effects are very similar to regular effects in that you apply them in the same way.

You use EffectVisualEffect and provide it the parameter of one of the visual effect constants... all of which start with VFX_*. There are a fair number of these, however, and they deserve some special mention.

VFX_BEAM_* = these are the different types of beam visuals, and while grouped with the other visual effects, these constants are used with EffectBeam and not with EffectVisualEffect.

VFX_COM_* = these are combat visual effects, almost all of which should be applied with an instant duration.

VFX_DUR_* = most of these are the visual effects used for spells. They all must be applied with either a permanent or temporary duration.

VFX_FNF_* = these are called 'fire-and-forget' effects in that they play only once. They must have an instant duration.

VFX_IMP_* = these are 'impact' effects, usually used for the short-duration effect when a spell strikes a target. They have an instant duration.

This is an example of a magic well which, when used, creates an effect on itself and then applies a strength increase with an effect to the target (but can only be used once by them... if they use it again it strikes them with lightning for 8d6 damage):

NWScript:

```
void main()
{
    effect eFirst;
    effect eSecond;
    effect eStrength = EffectAbilityIncrease(ABILITY_STRENGTH,
(d4() + 1));
    object oTarget = GetLastUsedBy();
    int nOnce = GetLocalInt(oTarget, "Use_Pool_Once");
    // if the PC has not used the pool before
    if (nOnce == FALSE)
    {
        eFirst = EffectVisualEffect(VFX_IMP_GOOD_HELP);
        eSecond =
EffectVisualEffect(VFX_IMP_IMPROVE_ABILITY_SCORE);
        // create the effect on the pool
        ApplyEffectToObject(DURATION_TYPE_INSTANT, eFirst,
OBJECT_SELF);
        // apply the strength effect with visual 3 seconds later
        DelayCommand(3.0,
ApplyEffectToObject(DURATION_TYPE_INSTANT, eSecond, oTarget));
        DelayCommand(3.0,
ApplyEffectToObject(DURATION_TYPE_TEMPORARY, eStrength, oTarget,
300.0));
        // and set the variable on the PC so they can't use the
pool again
        SetLocalInt(oTarget, "Use_Pool_Once", TRUE);
    }
    // if they have used the pool
    else
```

```

    {
        eFirst = EffectVisualeffect(VFX_IMP_LIGHTNING_M);
        // allow them a reflex save against the lightning bolt
        int nDamage = GetReflexAdjustedDamage(d6(8), oTarget, 25,
SAVING_THROW_TYPE_ELECTRICITY);
        eSecond = EffectDamage(nDamage, DAMAGE_TYPE_ELECTRICAL);
        // and then cause the visual effect of the bolt strike
and apply the damage
        ApplyEffectToObject(DURATION_TYPE_INSTANT, eFirst,
oTarget);
        ApplyEffectToObject(DURATION_TYPE_INSTANT, eSecond,
oTarget);
    }
}

```

Removing Effects

The thing to know about removing effects is that an effect has a unique 'pointer' established when it is applied. If you define a new effect variable later and define it as a similar type of effect, `RemoveEffect` will *not* target it.

The easiest way to remove an effect is to scroll through the current effects that are applied to a target through the use of `GetFirstEffect` and `GetNextEffect...` and then filter out the specific effect you are looking for. Once you have found it (or them if you are removing more than one effect at a time) the pointer is set to the correct effect and you can remove it.

The following commands are useful for filtering:

GetEffectCreator = if you want to specify only effects that were applied by specific creatures or objects
GetEffectDurationType = this will return the constant of an effect's duration type (either instant, permanent or temporary). Very general, but useful in specific circumstances.
GetEffectSpellId = this will return the constant of the spell that applied the effect (SPELL_*). Very useful if you are looking for all the effects of a particular spell or leaving out effects of certain spells.
GetEffectSubType = this will return the constants SUBTYPE_MAGICAL, SUBTYPE_SUPERNATURAL or SUBTPE_EXTRAORDINARY. All effects default to a magical sub-type unless specifically set otherwise.
GetEffectType = this is the most commonly-used filter when looking for a specific effect. It returns the constant EFFECT_TYPE_*, which almost duplicates the list of effect commands. Only visual effects don't have an effect type.

the basic script for removing an effect looks like this:

```

effect eEffect = GetFirstEffect(oTarget);
while (GetIsValidEffect(eEffect))
{
if (eEffect == filter used)
{
RemoveEffect(oTarget, eEffect);
}
eEffect = GetNextEffect(oTarget);
}

```

an example of removing a curse effect applied by a witch (with the tag "EVIL_WITCH") off of dialogue:

NWScript:

```

void main()
{
effect eEffect = GetFirstEffect(GetPCSpeaker());
while (GetIsEffectValid(eEffect))
{
if ((GetEffectType(eEffect) == EFFECT_TYPE_CURSE) &&
(GetEffectCreator(eEffect) ==
GetObjectByTag("EVIL_WITCH")))
{
RemoveEffect(GetPCSpeaker(), eEffect);
}
eEffect = GetNextEffect(GetPCSpeaker());
}
}

```

Some Things To Be Aware Of

- EffectKnockdown causes a crash when applied by anything other than a creature (will be fixed in next patch)
- EffectHitPointChangeWhenDying doesn't work (will be fixed in next patch)
- All effects are not permanent... they do not save with the character back to the vault, but will persist within the current module. The longest effect type currently is the extraordinary effect, which will resist dispel magic effects and resting. ExtraordinaryEffect and SupernaturalEffect are applied when declaring the effect command:

```
effect eEffect = ExtraordinaryEffect (EffectSilence());
```

- if there are any other hints or notices that you think should be included here, please let me know.

Celowin's Scripting Tutorial Lesson IV - UserDefined Events

Introduction

The purpose of this sequence of lessons is to take a complete beginner to programming, and teach him or her how to use NWScript to write modules. The early lessons will be very basic, and anyone that has done any coding at all will be able to skip over them. The goal here is to make the lessons so that even the people that just shudder at any type of code can learn.

Feel free to post these lessons on any forum, print them out, or modify them. However, just give me credit for doing them.

Any comments on these lessons, good or bad, can be sent to me at james.foxglove@verizon.net

I am going to assume that anyone looking at these lessons has at least played around with the Aurora Toolset a bit. If there is enough feedback that people don't know how to do the simple placements that I have in these lessons, I will consider spelling out in more detail what needs to be done.

For those of you that missed earlier lessons and are interested in looking them up, they have all been placed into the forum Scripting FAQ:
[Scripting FAQ](#)

Clean Up on Lesson Three

The more astute people following this sequence of lessons noticed that the guard script we came up with in Lesson Three was "good enough", but didn't behave exactly as we wanted it to. Every time the guard was "friendly", it would call out the greeting twice. Only people paying close attention would notice, but it is something that we should probably fix before going on. Besides, it gives me a chance to explain another concept.

First, we need to pinpoint why the guard was behaving that way... and it boils down to the OnPerception script handle. There are four things that can cause the attached script to fire:

- the npc sees something
- the npc hears something
- the npc notices something disappear from sight
- the npc notices something stop making sounds

So, what was happening is that the npc was both seeing and hearing the pc, and so the script was being executed twice. Not the end of the world, but not the behavior we wanted, either.

Let's fix this in the following way: We'll have the guard only react if it *sees* something. After all, the guard is looking for a ring, and those are usually pretty tough to hear.

We could do this by nesting another level of if statements, but it was already getting a bit confusing with all the different layers we had. Instead, what we want is for the script to check two things at once: we want to be sure the object perceived was a pc **and** that it was perceived by sight. Only if both things are true will we do all the rest.

We can do this just by modifying the condition, along with the operator "&&". &&, (read as simply "and") when put into a condition, is a way of linking two conditions together. The entire thing will be true only if both parts are true. To fix our script, then, all we have to do is change the line:

```
if (GetIsPC(oSeen))  
  
to  
  
if (GetIsPC(oSeen) && GetLastPerceptionSeen())
```

Then, the script will only run if it was a pc noticed by the guard, and also the guard *saw* the pc, instead of noticing the pc some other way.

The GetLastPerceptionSeen is a little function that returns TRUE if the last perception was by sight, FALSE if any of the other three.

I won't do an example right now, but another way of linking conditions together, instead of && is ||. || is read "or" and means that the condition is true if either one of the parts is true. This might be used if there were multiple reasons why the guard might attack. (Perhaps guard would attack if the pc didn't have the ring, or if the pc was carrying the head of the mayor. Either one by itself is a reason for the guard to attack.)

A Confession

Right now, I have to admit that I've been lying to you all throughout the past three lessons. Multiple times, I've said that I use the methodology I do is because I am doing everything the way I would inside a real module. But now I have to come clean – not a single script I have done here is really the way I would put it in final form.

The problem is, that our npcs so far have been totally unresponsive to most stimuli. The guard we finished up top is starting to get there, but I would hardly call its behavior realistic. If you're interested, here is an experiment you could do: Beef up the hp and level of the guard. Start up the module. Get the ring, so the guard will be friendly to you. Then go attack the guard. It will just stand there, and let you beat on it. This is definitely *not* what we want from most of our npcs.

The reason that we have been creating such morons is that we threw away all of BioWare's hard work in writing scripts. The default scripts that we have been deleting define a number of useful "standard" behaviors that we really probably want to keep in almost every instance. (Actually, one of the npcs we'll be creating today we *will* want to throw away the default scripts, but more on that when we get to it.)

So, how do we define our own behavior, without throwing away all the default stuff? We use the "user defined" script handle. If we are clever, we can really use every other handle – we'll make only minor modifications to the OnSpawn script, and do most of our scripting in the UserDefined.

We've spent so much time on our guard so far, let's go ahead and fix it up to the way it should be.

- Open up the Test Module
- Remove the guard
- Create a new npc where the guard was (So we have all the default scripts back)
- Change the npc tag to GUARD
- On the "Advanced" tab, make sure the guard has faction of "commoner."
- Go to the scripts tab
- Edit the "OnSpawn" script. There is a lot of stuff here, we'll ignore most of it.
- Go down to the bottom of the script. Find the line:

NWScript:

```
//SetSpawnInCondition(NW_FLAG_PERCIEVE_EVENT);    Fire User Defined  
//OPTIONAL BEHAVIOR                               Event 1002
```

- On that line, remove the first //. (Keep the second, in front of OPTIONAL)
- Save the script as tm_guard_os
- Close the script window, and go to the OnUserDefined handle.
- Select the script tm_guard_op
- Open it up into the editor. Save it as tm_guard_ud
- Update the comments to reflect that it is in a new place, and has a new name.
- Save again.
- Ok your way out of the guard, and save your module.

Now, if we start up the module, the guard will behave more realistically. He still does the

"friend or foe" reaction that we scripted into him, but also reacts to other stimuli. If you attack him, he fights back. There are lots of other behaviors that are scripted in there, many of which happen "behind the scenes" that you will probably never notice.

So, what did we actually do? Well, when removing the // in the OnSpawn script, we "uncommented" something. Remember that anything after // on a line in a script is ignored. So what BioWare did was put in a bunch of "optional" stuff into the OnSpawn script, and put // in front of it so that it wouldn't happen.

But now, we *want* some of it to happen. By removing the //, now we are saying that we want that line to actually take effect.

So, what does that line we "put back in" actually do? In essence, it is saying: "When you are running the OnPerception script, also do what I put into the OnUserDefined script."

Now, the more astute of you might be thinking ahead, and asking "What if I want to have multiple new behaviors from an npc? What if I want to have it do something special on OnPerceived and also on OnHeartbeat?"

It can still be done, but takes a bit of extra work. Just to keep the script small, let's make an npc that does something simple. It will say "I'm bored," every six seconds, and bow when it sees a pc.

- Open the toolset, paint the npc, and give it the tag BORED
- Open the OnSpawn script, and uncomment the lines for the OnHeartBeat and OnPerception lines.
- Notice that each of these has a "number" associated with it. 1002 for OnPerception, and 1001 for OnHeartBeat.
- Save it as tm_bored_os
- Put the following script into OnUserDefined, and save as tm_bored_ud

NWScript:

```
// On User Defined Script: tm_bored_ud
// Will be called by the OnHeartbeat and OnPerception scripts
//
// The npc will complain about being bored every six seconds,
// and will bow if it sees a pc.
//
int nCalledBy=GetUserDefinedEventNumber();
void main()
{
    switch(nCalledBy)
    {
        case 1001: // Called by OnHeartbeat
```

```

        ActionSpeakString("I'm bored.");
        break;
    //
case 1002: // Called by OnPerception
    object oSeen=GetLastPerceived();
    if (GetIsPC(oSeen) && GetLastPerceptionSeen())
        ActionPlayAnimation(ANIMATION_FIREFORGET_BOW);
    break;
}
}

```

Question and Answer time....

What is this GetUserDefinedEventNumber?

This is exactly the number I told you to pay attention to up there. BioWare was very clever... not only can each different handle call the user defined script, but each one passes a different number to it when it does so. So, what I'm doing in that first line is checking which one of the scripts called this one. Was it the Heartbeat (1001) or the Perception (1002)?

What about this switch command?

I don't want to go a whole lot into detail on this one. Basically, you give it an input of an integer, and the script then "jumps" to the line marked with "case" and that number. So if our nCalledBy is 1001, the script looks down for the line "case 1001:"

It starts doing commands at that point, until it gets to a break.

A few notes on formatting: The switch command only looks within the lines tied to it with { and }. Also, like the if statement, there is no semicolon after the switch line.

When I'm writing UserDefined events, I usually try to always set them up with a switch like this, even if I only plan on having one type.... just because I might change my mind later. It is better to be prepared for a possibility, than have to monkey with a bunch of work because you were too lazy to plan ahead.

Hey, you didn't use { and } with your if statement!

If there is only one command attached to the if statement, the curly braces aren't needed. To me, sometimes it looks better with them, and sometimes without them. I go with whichever makes the script look cleaner. In this case, I decided that they would just clutter up the script.

One More Example

At this point, you know a good deal about scripting. I shouldn't be calling these Absolute Beginner lessons any more, because you aren't. There are still a lot of functions you need

to learn, and a few more tricks. But really, there are lots of cool things that we can do with what we've learned so far, if we put the pieces together.

I'm going to do an example like that now. The script is pretty complicated, requires a lot of setup, and uses some new commands. But I think the end result is worth it.

- Open up the Test Module
- I want to get away from our guard, so go back to the first area that we created.
- Remove the SINGER npc that is there
- Paint in a module start point
- Paint a commoner npc in one corner of the room. (Doesn't really matter if it is a commoner, we're going to change pretty much everything.)
- On the "Basic" tab: Change the first name to Dartboard, the tag to DARTBRD, the race to "Construct", the appearance to Archery Target, the gender to none, and the portrait to po_PLC_F01_ (found by clicking the "Placeable Objects and Doors")
- On the "Advanced" tab, check the "Plot" box.
- Still on the Advanced tab, go to the faction editor. Create a new faction "Target", whose parent faction is "Hostile." Set both Target-Commoner and Commoner-Target to 50.
- Change the Dartboard faction to Target.
- On the "Scripts" tab, delete all the scripts. (In this case we *don't* want the dartboard going on a rampage and attacking people.)
- Edit the "OnDamaged" handle. Put in the following script.

NWScript:

```
// OnDamaged Script: tm_dartbrd_dm
// Dartboard script
// Goes "thunk" when hit with a ranged weapon.
//
void main()
{
    if (GetWeaponRanged(GetLastWeaponUsed(GetLastAttacker())))
    {
        SpeakString("***Thunk***");
    }
}
```

- If you want, you can test out just the dartboard by saving and loading up the module, but that isn't the cool part.
- Paint a waypoint about 1 square away from the dartboard.

- Give it the tag DARTWP001
- Paint a commoner npc near the waypoint.
- Change the npc tag to DARTPLAY
- For fun, click the "random name" button.
- Under the "Feats" tab, give it "Weapon Proficiency (simple)"
- Click the "Inventory" button, under the full body picture of the npc
- On the right, go to "Weapons", "Throwing", "Dart" and drag it to the "Contents" spot.
- Right click the dart you just dragged over, edit the properties, and change the stack size to 3
- Click OK, then click and drag the dart up to equip it on the npc.
- Click OK again, to get out of the inventory.
- Now, we go to the scripts tab.
- Edit the OnSpawn script, uncomment the HeartBeat line.
- There is another line to uncomment as well: SetSpawnInCondition (NW_FLAG_SET_WARNINGS);
- Also, add in a line somewhere: SetLocalInt(OBJECT_SELF, "DARTSTATE", 1);
- Save the script as tm_dartplay_os
- Now go to the UserDefined handle, and add in the following script:

NWScript:

```
// On User Defined Script
// tm_dartplay_ud
// Used to have someone play darts. Called by the OnHeartBeat
script.
//
// The Dart Player will throw all darts in inventory (should
start with 3),
// walk to the dartboard, get 3 darts, walk back, and repeat.
//
void main()
{
    int nCalledBy = GetUserDefinedEventNumber();
    object oTarget = GetNearestObjectByTag("DARTBRD");
    int nDartsReady = GetLocalInt(OBJECT_SELF, "DARTSTATE");
//
    switch(nCalledBy)
    {
        case 1001: // Called by OnHeartbeat
            //
            // nDartsReady will be 1 if ready to throw, 2 if not.
            //
            if
                ((GetIsObjectValid(GetItemInSlot(INVENTORY_SLOT_RIGHTHAND)))
                 && (nDartsReady == 1))
            {
                // If we have darts in the right hand, and we're
ready to throw, go for it.
            }
        }
    }
}
```

```

        ClearAllActions();
        ActionAttack(oTarget, TRUE);
    }
    else
    {
        // Otherwise, there are two cases. We've either
just run out, or we are in
        // the process of getting darts. We don't want to
interrupt the cycle if we're
        // already working on it.
        if (nDartsReady == 1)
        {
            SetLocalInt(OBJECT_SELF, "DARTSTATE", 2);
            ActionMoveToObject(oTarget);
            ActionWait(0.5);
            ActionPlayAnimation(ANIMATION_LOOPING_GET_MID,
1.0, 1.0);
            ActionWait(0.5);
            ActionPlayAnimation(ANIMATION_LOOPING_GET_MID,
1.0, 1.0);
            ActionWait(0.5);
            ActionPlayAnimation(ANIMATION_LOOPING_GET_MID,
1.0, 1.0);
            object
oDestination=GetNearestObjectByTag("DARTWP001");
            ActionMoveToObject(oDestination);
            CreateItemOnObject("nw_wthdt001", OBJECT_SELF,
3);
            ActionEquipMostDamagingRanged();
            ActionDoCommand(SetLocalInt(OBJECT_SELF,
"DARTSTATE", 1));
        }
    }
    break;
}
}
}

```

Well, this is a really complicated script. I'm not going to explain absolutely everything about it, but I'll point out a few key things.

I'm using the local variable DARTSTATE to make sure that we don't attack when not ready, or go through the "get darts" sequence multiple times.

The ActionDoCommand is **amazingly** handy. It takes a non-queued command, and turns it into a queued one. Normally, as soon as the script sees a SetLocalInt instruction, it sets the local variable. This forces the script to wait until the npc has completed all the previous actions.

The CreateItemOnObject is what is used to make the new darts for the npc. "nw_wthdt001" is the blueprint ref for a dart, and the final 3 is the stack size.

Other than that, see if you can trace through the script yourself. There are other new

commands, but most of the names are pretty self explanatory. As always, ask questions if you can't figure it out.

Coming Lessons

I haven't decided on exactly what order I want to do things in yet. It might be a few days until I get another lesson out. However, here are a few things that I want to talk about:

- Other script handles
- Scripting for objects
- Scripting in conversations
- Other things as I think of them, and as people suggest them

USING DIALOGUE WITH PLACEABLES AND TRIGGERS

There are lots of effects that a savvy DM can simulate just by using a bit of dialogue in conjunction with a placeable or trigger. You can have ambient descriptions of an area or object appear, start a 'dialogue' with a strange object that allows the player options of interacting with it, have someone on the other side of a door speak through it, inform players of events that occur... there are lots of possibilities.

The first thing you should know is what your options are. The following are commands that can be used to initiate dialogue or display text:

ActionSpeakString and SpeakString

The only difference between these two actions is that ActionSpeakString puts the command into the action queue (meaning it will not be performed until previous actions in the queue, such as ActionWait or ActionMoveToObject, have been completed) and SpeakString is performed immediately.

What it does: causes a line of text to be displayed over the performer of the command. This is displayed to the player as normal dialogue, the text appearing beside the name of the object and its portrait (if applicable).

Advantages: A "volume" can be applied to the text by placing the appropriate TALKVOLUME_* constant in the command. TALKVOLUME_TALK is the default and is heard by all at normal range. TALKVOLUME_SHOUT is heard by all in the module. TALKVOLUME_WHISPER is heard by all at close range.

TALKVOLUME_SILENT_TALK and TALKVOLUME_SILENT_SHOUT are variants on the normal talk and shout... but they will *only* be displayed to and heard by non-players.

Part of having a volume is that the text is 'heard' by objects in the game. Creatures within hearing range will turn their heads to notice, and if you are doing any text-recognition scripting via SetListenPattern and GetListenPatternNumber (such as combat 'shouts'), SpeakString can be used for that purpose.

ActionStartConversation

What it does: the commands the performing object to initiate dialogue with the target. It will set off the target's OnDialogue event (if they have one, which PC's do not). If no dialogue file resref is specified, the one attached to the creature or object will be used (the one you specified in the toolset). The token in the conversation will always be the target object.

Advantages: You can start a full conversation this way, with response choices and multiple starting conditions and scripts running off of it. You can also specify in the main command whether or not this is a private conversation... if it is, then only the target will see the text displayed on-screen (great for interacting with objects, since other players wouldn't 'hear' this conversation).

BeginConversation

What it does: this command is normally only used in the OnDialogue event of creatures. This is the command that tells them to start dialogue with the player that just clicked on them. Using it on an object does not set off their OnDialogue event. If the target of the

action is not specified, the object that initiated their OnDialogue event will be used.
Advantages: Can be used for NPC to NPC conversations... otherwise
ActionStartConversation should always be used outside of the OnDialogue event.

FloatingTextStringOnCreature

What it does: when you perform many actions in the game, you will notice that the text that appears over the player's head floats upward and fades away. This command does the same thing. The text also shows in the player's chat display... it does not show as originating from any creature, nor does it show any portrait. The text simply displays in yellow, as most ambient text does.

Advantages: great for ambient text or simulating a written display of an action, since it does not prefix the text with a name or even look like dialogue in the display.

One Thing to Be Aware of: Floating text, by default, is ONLY able to be seen by the creature that it is floating over. If you set the command 'display to faction', then members of the same faction will also see the text displayed. This does mean that floating text is really only useful on PC creatures, since no non-PC creature or object can be set to the PC faction.

SpeakOneLinerConversation

What it does: it acts very much like ActionStartConversation, but its purpose is more specific. It is meant to access the dialogue file, but only for a single line with no responses or links.

Advantages: You can still script the starting conditions for the dialogue file, so you can have a whole selection of single-liners that the speaker can choose from... and you can still use action scripts off of those lines. The speaker does not turn to any target before speaking... and the object identified in the command (if one is) becomes the reference for any tokens used in the dialogue (even if they are not present).

Making A Placeable Object Display Text

For the example I'm going to use here, we're going to make a placeable object signpost display text for the player that clicks on it.

****IMPORTANT!**** If you are going to have any placeable object interact with others in this way, make sure the 'Static' box is unchecked! If a placeable object is static, it is intended to be merely part of the background art and unselectable. Most placeable objects are marked static by default!

So select a signpost from the placeable object list and place it down. Then go into its properties and make sure 'Static' is unselected. Since I want the player to be able to click on the signpost, check the 'Useable' box.

Go into its script section and put the following script under the OnUsed event:

NWScript:

```
void main()
{
    SpeakString("North: Waterdeep, 20 miles");
}
```

Now when you are in the game and click on the signpost, it will display that text next to its portrait (which is, by default, a signpost) and name. Naturally you can always change the portrait to whatever you like, as well as the name, and the display will be different.

Making a Trigger Display Floating Text

In this example, we want it so that when a player crosses into a laid-down trigger, a piece of ambient text appears over their head.

Go into the trigger menu and select 'Generic Trigger'. This will allow you to draw the bounds of the trigger box. If I wanted to display a description of something the player sees in a room, for example, I might put this just inside a door.

Then go to the scripts section of the trigger. Since we want the text to appear when the player enters the trigger, we'll put the script in the OnEnter event. Some things to consider: do we want it to display only for PC's, or every entering creature? Do we want it to display every time a PC enters, or only once? For now, I'll have the text appear on anyone who enters the trigger... but only the first time.

NWScript:

```
void main()
{
    object oPC = GetEnteringObject();
    if (GetLocalInt(oPC, "Saw_Trigger_Text") == FALSE)
    {
        FloatingTextStringOnCreature("The room is dark and
spooky.", oPC);
        SetLocalInt(oPC, "Saw_Trigger_Text", TRUE);
    }
}
```

When the player steps into the trigger in-game, the text will appear over their heads and float up, fading. It will also display in their chat display window in yellow.

Note that triggers can't use SpeakString or ActionSpeakString. They simply can't speak, period. It can, however, tell another object to speak... whether that object be an invisible object right on it or elsewhere. If I had a nearby placeable object with the tag

"DISPLAY_HERE" (which had its 'Static' box unchecked, remember) and I wanted the trigger to make it display the above text, I would use this script:

NWScript:

```
void main()
{
    object oPC = GetEnteringObject();
    object oDisplay = GetObjectByTag("DISPLAY_HERE");
    if (GetLocalInt(oPC, "Saw_Trigger_Text") == FALSE)
    {
        AssignCommand(oDisplay, SpeakString("The room is dark and
spooky."));
        SetLocalInt(oPC, "Saw_Trigger_Text", TRUE);
    }
}
```

Having an Object Display Text When a PC Comes Near

In this example, I'm going to lay down an invisible object (that the players can neither see, highlight nor touch) that is going to display text when a player comes within 10 meters.

First, select the Invisible Object from the placeable object menu and lay it down. Go into its properties and make sure 'Static' is unchecked. I don't want the player to be able to click on it, so leave 'Useable' unchecked. Change the name of the object to 'Unknown Person' and leave the portrait blank.

Go into its scripts section. Put the following script into its OnHeartbeat event:

NWScript:

```
void main()
{
    object oPC = GetNearestCreature(CREATURE_TYPE_PLAYER_CHAR,
PLAYER_CHAR_IS_PC);
    if (GetIsObjectValid(oPC) && (GetDistanceToObject(oPC) <
10.0))
    {
        SpeakString("Psst! Come over here!");
    }
}
```

In the game, when the player gets within 10 meters of the object, the text will be displayed on-screen... and in the chat display window it will read "Unknown Person: Psst! Come over here!". That text will repeat every 6 seconds (every heartbeat) that the PC remains within range.

I could further develop the script by adding to the above script... say, checking to see if the player comes within 3 meters and spawning in the 'appearing' Unknown Person on top of the placeable object.

One thing you might notice is that the placeable object is pretty small... and text displaying over it appears pretty low to the ground (depending on how zoomed-in the camera is). Here's a trick to place it higher: before placing the invisible object, select another placeable object that is about as high as you need the invisible object to be. Place it, first. Then when you select the invisible object from the menu, place the cursor directly 'over' the first object before clicking to set it down. It will appear on top of the first object. Now delete the first object and your invisible placeable floats in mid-air (and will stay there).

Having a Placeable Object Start a Dialogue

Let's say that I want an object that the player can select and get an interactive dialogue... or perhaps the object actually speaks.

First, go into the placeable objects list and pick out an object. Select a pedestal, for instance, and place it in the module. Go into its properties and ensure that 'Static' is unchecked and 'Useable' is checked.

Go to the advanced tab. Write in a name for the dialogue file we are going to create and press Edit. You can start with a line like "There is some fine writing on the pedestal which is in a strange language. Do you try to decipher it?" You could then continue to give responses and make the dialogue as long as you like, just as normal dialogue.

Go into the scripts tab and place the following script under OnUsed:

NWScript:

```
void main()  
{  
    object oPC = GetLastUsedBy();  
    ActionStartConversation(oPC, "", TRUE);  
}
```

In the game, when the player clicks on the pedestal, he will approach it and the dialogue will ensue normally. Since I specified TRUE in the bPrivateConversation parameter of the command, only he will see the dialogue... another player standing next to him would see and hear nothing.

If I didn't care whether the text displayed, I could simply have written ActionStartConversation(oPC), since the "" just means that the attached dialogue file is used.

Starting a Conversation With a Door

Let's say I want to simulate this: the player selects a door to open but finds it locked. A voice speaks from the other side: "What's the password?"

First, place the door normally. Go into its properties and change the name. The great thing about a door's name is that it doesn't show up anywhere (even on Examine) except in dialogue... so let's change it to "Guard on other side", since that's what the player will see.

Mark the door as 'plot' so it can't be broken down (although you could certainly let it be, and even script consequences for attempting to bash in the OnDamaged or OnPhysicalAttacked events). Go into the lock tab and check the 'Locked' box. If you don't want it pickable, check the 'Key required to lock or unlock' box (although, again, you can leave it if you wish).

Go into the advanced tab and put a name into conversation and press Edit. We can make the first line "What's the password?" and continue as we wish... whenever the point in the dialogue is reached where the guard agrees to open the door, place this script in that node's 'Actions Taken' section:

NWScript:

```
void main()
{
    ActionDoCommand(SetLocked(OBJECT_SELF, FALSE));
    ActionOpenDoor(OBJECT_SELF);
}
```

Now we have to place the script that starts the conversation. Place the following script in the OnFailToOpen event:

NWScript:

```
void main()
{
    object oPC = GetClickingObject();
    if (!IsInConversation(OBJECT_SELF))
    {
        ActionStartConversation(oPC);
    }
}
```

In the game, then, the player will click on the door and approach it. He will get the 'door rattle' sound when the door fails to open, and suddenly the dialogue will appear. He will

see the portrait of the door and 'Guard on other side' at the top... with the text "What's the password?" displayed. When the dialogue point is reached for the door to open, it is unlocked and swings open... the player can now use the transition beyond.

Those are a few examples of using text and dialogue with objects. Hopefully they will be of some use to you.

[Edited By David Gaider: Monday, 15 July 07:42AM (GMT)]

ERFs : WHAT THEY ARE AND WHAT THEY'RE GOOD FOR by Darmoor Dragon

.ERF Files - Drag and Drop NWN for non-scriptors?

The NWN Toolset provides an excellent feature for importing and exporting certain elements from within one Module to another in a format of .erf (exported resource file?).

This allows Module makers to export elements such as conversations, scripts, entire areas or the "blueprints" for placeables, items, creatures, sounds, triggers and more!

Why are .ERFs a good thing?

ERFs export from a module with "everything attached" - as an example if you exported a chair that had a custom script on it that allows a player to sit on it, then not only will the chair be exported but it will **include** the custom script

placed within the chair in its appropriate script location.

This means that if you import this into any other module you can simply select the chair and place it as a custom object and it will already be a "working sitable chair". Similarly NPCs exported with custom scripts will export with **all non-default scripts** attached to them (including conversation files).

One distinct advantage in this is that you (the end user) need not worry about messing up a script within/on a creature or placeable, putting a script in the wrong event, misnaming a script and so on: ERFs are largely self-contained and where they don't work its probable they never did, or, some extra element is missing.

Some examples:

A working Smith:

I create a working smith that hammers on an anvil, complete with sounds, special effects and a conversation tree and stops working whenever a Player comes within a trigger (thus allowing the smith to be made into a merchant).

As a scripting exercise this can be quite complex - as an ERF it is simple:

The creator of the Smith simply exports 3 ERFs: 1 for the smith "creature", 1 for the "anvil" and one for the trigger. The "user" simply imports the 3 files and places them. That's it! Working smith, no scripting required...well, not by the end user anyway.

A working Ballista:

..that fires AoE Acid bolts... Import just the one "Acid Bolt Ballista" .erf and that's it. Place this custom placeable in your module and it works...done.

A Woodcutter and regrowing trees:

3 .erfs: Woodcutter, stool and a "chopable tree".. once again place all 3 objects and it works, a living breathing (and singing) lumberjack who cuts down trees that grow back afterwards.

ERFs - how to:

ERFs work by being added to the Custom Palette for the module, importing ERFs automatically adds those elements to the Custom Palette but to Export them you must first add the element to the palette.

Once again this is very simple: simple right-click on the element to add and you will see an option to "Add to Palette" - once done you can export it.

When Exporting the ERF you can also add in text information on how to use the ERF. This information is displayed to the Importer when they import the ERF and any instructions and warnings on use should be placed in this area.

Importing is equally simple - Open the module you want to import the ERF into, and go File -> Import, then browse to the .erf file you want to import. The creators user information and guidelines (as detailed above) will show up and the file will be added to your own modules palette. That's it.

Important points:

Using ERFs is fundamentally simple and excellent for the distribution of module elements - Especially to the more scripto-challenged of us (aka: myself). Having said that there are some points that need to be watched **very** carefully.

Naming conventions and Import/Export issues:

When you import an ERF it brings in the attached scripts it uses "in the background" but it does tell the user if any name conflicts happen with existing script/file names in the module. Choosing to overwrite existing names can cause your module to experience problems.

Because of this it is important for those who export ERFs to use specific filename on any attached scripts and files such as conversations. e.g don't save a "smith" script as "smith" but use something like JMN_smith (your initials/module initials_smith) or similar.

This makes it far less likely that you will overwrite any existing script names in the end-users module.

For the "End-User":

When importing ERFs if you get a warning of any filename conflicts do NOT overwrite them - instead make a note of the filenames and cancel the importation. To get around such issues simply import the ERF into a new "empty" module and rename the files so they won't conflict (obviously this may require editing relevant script functions in the elements - no way around that sorry).

Distributing ERFs

Ideally ERFs should be "packaged" within a Zip file including a readme on how to use the elements within them. Those "packages" with multiple ERFs (such as the smith, anvil and trigger) require all elements to work as intended, how to place them and edit them should be included in the readme file.

Obviously ERFs need to be downloaded or emailed to the end-user, so you're going to need to host these files somewhere. For those without their own webspace, sites such as NWN Vault will hopefully start to host them (once ERFs are a little more commonly used and understood) their file sizes are tiny; barely larger than their component scripts, and emailing them as attachments is also an option.

Where to get existing ERFs?

Gulbsoft.de already have some ERFs available : [Click Here](#)

Alternatively wherever you see people posting about working elements within their modules - or scriptors who have a working element that you would like, rather than having to mess around with creating your own and editing scripts , then ask them to provide ERFs.

If they have a working example in their module it really is little more effort than a right-click, add to palette and Export File.

Happy ERFing!

FACTIONS, SHOUTS AND ATTACKING MY ENEMY

Just what is a faction?

A faction is one of three things: a party of player characters & their associates, a single player character who is not a member of a party & his associates or a group of NPC's or single NPC's that have been assigned to a set default relationship to other factions.

This is *not* to be mistaken with an 'organization', like a guild of thieves or a city's garrison. Players do not 'join' factions, and any organization could conceivably comprise of a number of factions.

What do factions do, then?

Just what they are supposed to do: set the default stance of the NPC towards other factions. The generic AI takes a stance of 'Friendly' or 'Hostile' and has the NPC react accordingly in its scripts. An NPC will maintain that default stance until and unless it is told to do otherwise.

What is probably most important to understand about factions, however, is that there is a difference between how the faction feels and how the individual NPC feels. If an individual NPC within a faction is attacked, he will go hostile and defend himself. He will remain hostile until told to do otherwise. The other members of his faction not present, however, will not *necessarily* go hostile along with him. This difference of standing towards someone is considered *reputation*, which is different from that someone's reputation with the faction. The NWScript commands, unfortunately, refer to both types of reputation in the same manner, so distinguishing between them can be tricky.

To better explain the difference, consider this:

A party of PC's is inside a tavern. The rogue in the party decides to attack a nearby peasant (who is a member of the Commoner faction). Upon being attacked, that peasant goes hostile to the rogue... and issues a shout to any nearby NPC's who are friendly to his faction to likewise go hostile to the rogue. The rest of the party hasn't done anything, yet, so nobody is hostile towards them.

If the rest of the party gets involved in the fight, the witnessing NPC's will likewise go hostile towards them. Let's say the entire party flees the tavern immediately and escapes their pursuers... those individuals who turned hostile towards the rogue (or other interfering party members) will remain hostile until told otherwise. Other members of the Commoner faction, however, will react normally.

The only way this *wouldn't* happen is when a faction has its 'Global Effect' box checked (this can be found in the Faction Editor). This eliminates individuals within a faction maintaining separate relations... if you attack one and it goes hostile, all members of the faction go hostile. Any change to a single member's relationship affects the entire faction. (There are instances in module design where this is desirable... but be aware of it,

especially due to the fact that new factions are global in effect by default.)

What does the standard AI do with factions?

To better understand why something is or isn't happening, here's a rundown on how factions affect the standard AI:

- in the OnPerception event (which, remember, fires only when the target first comes into perception range or if the perception type changes), if the NPC perceives a hostile and is not already in combat it issues the "NW_I_WAS_ATTACKED" shout (see below) and attacks.
- if the creature is attacked and it is not already in combat, it issues the "NW_I_WAS_ATTACKED" and "NW_ATTACK_MY_TARGET" shouts and attacks its attacker.
- if the creature is killed it automatically issues a "NW_ATTACK_MY_TARGET" and "NW_I_AM_DEAD" shout.

And what do these shouts do? For one thing, they are simply spoken words that the creature has been told to speak, the same as any PC typing words with their chat system. These shouts, however, are 'silent'... players don't hear them or see the floating text, but NPC's do, and the generic AI in the OnSpawn script tells them to listen for them and react. Shouts are not heard through walls or beyond doors, and do have a limited range.

- the "NW_ATTACK_MY_TARGET" shout only has an effect on those NPC's who have uncommented the line 'SetSpawnInCondition (NW_FLAG_SHOUT_ATTACK_MY_TARGET);' in their OnSpawn script. If heard by an ally (one who's faction is friendly to the shouter), it sets the faction relationship to the shouter's attacker to hostile (if not already so), stops what it's doing and attacks any enemy within sight.
- the "NW_I_WAS_ATTACKED" shout can be heard by anyone. If the listener is an ally of the shouter and is neither currently in combat nor has less than 10 Commoner levels, it will attack the attacker of the shouter.
- the "NW_I_AM_DEAD" shout can also be heard by anyone. It's effect is the same as the 'I was attacked' shout above.

HOW DO I MAKE MY NPC'S ATTACK?

If a creature starts out as hostile and spots a PC, its OnPerception event will fire and it will immediately attack. No problem there. The situation arises fairly often, however, where you will have a NPC who was neutral to the PC when the OnPerception event fired and will become hostile through the course of a dialogue or scripting.

To turn a single NPC hostile to the PC and make him attack, you must first identify the target. In dialogue, this is easy:

```
object oTarget = GetPCSpeaker();
```

Outside of dialogue, in a normal script, it can vary:

```
object oTarget = GetNearestCreature (CREATURE_TYPE_PLAYER_CHAR,  
PLAYER_CHAR_IS_PC);  
// this targets the nearest player character to the caller of the  
script
```

```
object oTarget = GetLastPerceived();  
// this is the last perceived object, only to be used inside the  
OnPerception event
```

The above are two possibilities. Regardless, once you've identified your target, you can call the rest of the commands as normal. For the following scripts, we'll assume that we're starting combat off of dialogue (by placing the script in the 'Actions Taken' section of the dialogue line where we want combat to start).

First, do I want the attacker to turn the whole faction hostile? To do so, I use this:

NWScript:

```
#include "NW_I0_GENERIC"  
void main()  
{  
    // lower my faction's relationship with the PC by 100 (to  
    hostile)  
    AdjustReputation(GetPCSpeaker(), OBJECT_SELF, -100);  
    // start combat (this requires the nw_i0_generic file to be  
    included, above)  
    DetermineCombatRound();  
}
```

And that's it (this is also the "nw_d1_attonend" generic script). Should I just want the attacker to go hostile (and no-one else):

NWScript:

```
#include "NW_I0_GENERIC"  
void main()
```

```

{
    // set myself to hostile against the PC
    // note that the 'temporary' is 3 minutes in length unless a
specific
    // amount of time is supplied in the command.
    SetIsTemporaryEnemy (GetPCSpeaker());
    // start combat (this requires the nw_i0_generic file to be
included, above)
    DetermineCombatRound();
}

```

But suppose I have other friends nearby who I want to make part of the combat, as well? Even if I use AdjustReputation to make their faction hostile, they still won't attack. Why? Look at the cases under the generic AI... I wasn't attacked first and am already in combat, so I won't issue those shouts. We've all already perceived the PC, and he wasn't an enemy then. And I'm not dead... so my pals will sit there until attacked or until I'm dead. I've got two choices. Either send them the command via AssignCommand like this:

NWScript:

```

#include "NW_I0_GENERIC"
void main()
{
    // this script works great if I know exactly who I want to
involve in the fight
    // and they have unique resrefs
    object oGoblin2 = GetObjectByTag ("GOBLIN2");
    object oGoblin3 = GetObjectByTag ("GOBLIN3");
    // lower my faction's relationship with the PC by 100 (to
hostile)
    AdjustReputation (GetPCSpeaker(), OBJECT_SELF, -100);
    // tell my friends to start fighting
    AssignCommand (oGoblin2, DetermineCombatRound());
    AssignCommand (oGoblin3, DetermineCombatRound());
    // start combat (this requires the nw_i0_generic file to be
included, above)
    DetermineCombatRound();
}

```

Or like this:

NWScript:

```

#include "NW_I0_GENERIC"
void main()
{
    // this script will make every ally within the current area
attack
    location oHere = GetLocation (OBJECT_SELF);
    // lower my faction's relationship with the PC by 100 (to
hostile)

```

```

AdjustReputation (GetPCSpeaker(), OBJECT_SELF, -100);
// now cycle through all objects in my area
object oFriend = GetFirstObjectInArea (GetArea(OBJECT_SELF));
while (GetIsObjectValid(oFriend))
{
    // if the object is a creature that is a member of my
faction
    if (GetFactionEqual(oFriend) && (GetObjectTypes(oFriend)
== OBJECT_TYPE_CREATURE))
    {
        // and it can see the PC
        if (GetObjectSeen (GetPCSpeaker(), oFriend))
        {
            // tell him to start combat
            AssignCommand (oFriend, DetermineCombatRound());
        }
        // otherwise, if he can't see the PC
        else
        {
            // tell him to stop what he's doing
            AssignCommand (oFriend, ClearAllActions());
            // and come to my location
            AssignCommand (oFriend,
ActionMoveToLocation(oHere, TRUE));
        }
    }
    oFriend = GetNextObjectInArea (GetArea(OBJECT_SELF));
}
// now start fighting, myself
DetermineCombatRound();
}

```

There are also other variations to do the above... those are really just examples. Another easy way to start a group fight is to make sure all the combatants have the 'SetSpawnInCondition (NW_FLAG_SHOUT_ATTACK_MY_TARGET);' line uncommented in their OnSpawn script. Then, when you start the attack, you can just do this:

NWScript:

```

#include "NW_I0_GENERIC"
void main()
{
    // lower my faction's relationship with the PC by 100 (to
hostile)
    AdjustReputation (GetPCSpeaker(), OBJECT_SELF, -100);
    // start combat (this requires the nw_i0_generic file to be
included, above)
    DetermineCombatRound();
    // and shout to my allies in the vicinity to join me
    SpeakString ("NW_ATTACK_MY_TARGET", TALKVOLUME_SILENT_TALK);
}

```


You won't see the string displayed, but any allies within hearing range who can see the target will run to the attack.

NOW HOW DO I MAKE MY NPC'S NEUTRAL AGAIN?

If you want to make an NPC non-hostile once it's been in combat with the PC (usually if the PC has died or for plot reasons), you have to make sure two things have happened: one, that the faction's attitude towards the player is non-hostile (either neutral or friendly). Two, if the faction doesn't have 'Global Effect' checked, you have to clear the NPC's personal feelings towards the PC.

The 'standard' factions in the game (Commoner, Defender, etc) are the only ones that can do both of these in one single command. This is done through the SetStandardFaction command, and an example of this is in the default death script ("nw_o0_death"), which re-sets all standard factions to neutrality on a player's death. (*NOTE*: Several people that have reported that the Defender faction is not re-set when this command is used, but can be re-set if treated as a custom faction below.)

For custom factions, you need two commands: AdjustReputation to change the faction's relationship with the target and ClearPersonalReputation to change an NPC's feelings. In both cases, you must single out a target that is a member of a faction (which may be difficult if you have many members with separate tags or that may be dead)... custom factions aren't identified by name, you can only say 'the faction that so-and-so belongs to'. And if you want to clear the a PC's personal reputation with an *entire* custom faction, you have to cycle through it.

Here's an example script that will adjust a faction towards neutral and cycle through it, restoring every member to a normal relationship with the PC. This faction does not have its 'Global Effect' checked, obviously, and has members with the same tag "GOBLIN01". The PC has died and this script is within his OnDeath event:

NWScript:

```
// I'm creating a custom 'ClearAll FactionMembers' command, here
// first declaring the api for the new command, then what it does
// I could reasonably have put this in a separate script and
// used #include to put it into this script (and others), as well
void ClearAllFactionMembers (object oMember, object oPlayer)
{
    object oClear = GetFirstFactionMember (oMember, FALSE);
    while (GetIsValidObject(oClear) == TRUE)
    {
        ClearPersonalReputation (oPlayer, oClear);
        oClear = GetNextFactionMember (oMember, FALSE);
    }
}
```

```

// here's the main body of my script
void main()
{
    // identify the player
    object oPlayer = GetLastPlayerDied();
    // identify a member of the faction. I'm assuming these
members are alive.
    // Otherwise, I would have to try several things to turn up a
member that is
    // first valid and then do the if command
    object oGoblin = GetObjectByTag("GOBLIN1");
    if (GetIsObjectValid(oGoblin))
    {
        // adjust the faction relation back up by 100
        AdjustReputation (oPlayer, oGoblin, 100);
        // run my custom command to cycle through the faction
        ClearAllFactionMembers (oGoblin, oPlayer);
    }
}
}

```

FUN WITH SHOUTS

A couple of things to try out or use, with regards to shouts.

First one: I always hate the fact that an NPC may hear the "NW_ATTACK_MY_TARGET" shout... but they won't do anything if they can't see the shouter or the target. They go hostile, but won't attack until the enemy comes into sight. Here's a way I've successfully used to make out-of-sight listeners come to investigate the shout.

step 1 - First thing that must be done is that all members of the faction who will respond to the shout and also investigate a shout from nearby must have their OnSpawn script set up accordingly. Go into their OnSpawn script and uncomment the following lines:

```
'SetSpawnInCondition (NW_FLAG_SHOUT_ATTACK_MY_TARGET);'
'SetSpawnInCondition (NW_FLAG_ON_DIALOGUE_EVENT);'
```

Then re-save the script under a different name.

step 2 - Remember that a creature will only shout 'attack my target' if he is attacked first or killed. If you want him to shout it, himself, you'll have to put it into a script. There is one supplied in the examples above that has the creature go hostile and make the shout off of dialogue.

step 3 - Put the following script into each creature's OnUserDefined event:

NWScript:

```
// this script will make the receiver of a "NW_ATTACK_MY_TARGET"
shout come and
```

```

// investigate even if he doesn't see an enemy. Once he
approaches the shouter,
// if he sees an enemy his OnPerception script will activate and
he will attack.
// An interesting modification to this script would be to have
the out-of-sight
// receiver of the shout issue a second shout which would carry
the original shout
// further... this would, however, require a new shout and new
pattern definition
void main()
{
    int nEvent = GetUserDefinedEventNumber();
    if (nEvent == 1004) // OnDialog event
    {
        // this sees if the shout matches the pattern set by the
'SetListeningPatterns' in OnSpawn
        int nMatch = GetListenPatternNumber();
        object oShouter = GetLastSpeaker();
        object oIntruder;
        // if I recognize the shout and the shouter is a valid,
friendly NPC
        if(nMatch != -1 && GetIsObjectValid (oShouter) &&
!GetIsPC(oShouter) && GetIsFriend (oShouter))
        {
            // and the shout is "NW_ATTACK_MY_TARGET"
            if (nMatch == 5)
            {
                // attempt to define the shouter's enemy
                oIntruder = GetLastHostileActor (oShouter);
                if(!GetIsObjectValid (oIntruder))
                {
                    oIntruder = GetAttemptedAttackTarget();
                    if(!GetIsObjectValid (oIntruder))
                    {
                        oIntruder = GetAttemptedSpellTarget();
                        if(!GetIsObjectValid (oIntruder))
                        {
                            oIntruder = OBJECT_INVALID;
                        }
                    }
                }
            }
            // if I can see neither the shouter nor the enemy
            if (GetIsObjectValid (oShouter) && !GetObjectSeen
(oIntruder) && !GetObjectSeen (oShouter))
            {
                // define the location of the shouter
                location lShouter = GetLocation(oShouter);
                // stop what I am doing
                ClearAllActions();
                // and move to that location
                ActionMoveToLocation (lShouter, TRUE);
            }
            // otherwise, if I can see the shouter but not
the enemy
            else if (GetIsObjectValid (oShouter) &&
!GetObjectSeen (oIntruder) && GetObjectSeen (oShouter))

```

```

        {
            // stop what I am doing
            ClearAllActions();
            // and move towards the shouter
            ActionMoveToObject (oShouter, TRUE);
        }
    }
}

```

second one: Here's an interesting exercise that demonstrates how to make your own shout. In this example, I want to have a dialogue that issues a shout and causes all who hear it to perform the 'worship' animation (they think the PC is the 'one foretold').

step 1 - Let's set up the dialogue of the NPC who is going to issue the shout. Let's make it very simple... one line of dialogue which says "It is the One Foretold!!" and then the following script attached to the 'Actions Taken' area of that line:

NWScript:

```

void main()
{
    // set a variable on myself equal to the PC I'm talking to
    SetLocalObject (OBJECT_SELF, "worship", GetPCSpeaker());
    // issue the "BOWDOWN" shout
    SpeakString ("BOWDOWN", TALKVOLUME_SILENT_TALK);
    // perform the worship animation for 30 seconds
    ActionPlayAnimation (ANIMATION_LOOPING_WORSHIP, 0.5, 30.0);
}

```

step 2 - we have to set up the other NPC's in the area to listen to the shout. This involves uncommenting the line 'SetSpawnInCondition (NW_FLAG_ON_DIALOGUE_EVENT);' in their OnSpawn script as well as adding the following line anywhere:

```

SetListenPattern (OBJECT_SELF, "BOWDOWN", 100);

```

This makes the string 'BOWDOWN' equal to the number 100 (picked at random) and recognizable should the NPC be listening (using the command SetListening (OBJECT_SELF, TRUE)... which is issues in the OnSpawn's SetListeningPatterns command, anyway).

step 3 - now we just have to tell the NPC's what to do when they hear that command. Any spoken string sets off their OnDialogue event... so the following script can be placed into their OnUserDefined event:

NWScript:

```

void main()

```

```

{
    int nEvent = GetUserDefinedEventNumber();
    // if I receive the OnDialogue event
    if (nEvent == 1004)
    {
        // see if I heard a matching pattern
        int nMatch = GetListenPatternNumber();
        // identify who spoke the string
        object oShouter = GetLastSpeaker();
        // and grab the variable from the NPC as to who they're
        bowing down to
        object oWorship = GetLocalObject(oShouter, "worship");
        // if the string is one I recognize and the speaker is a
        friendly, valid NPC
        if(nMatch != -1 && GetIsObjectValid (oShouter) &&
        !GetIsPC (oShouter) && GetIsFriend (oShouter))
        {
            // and the string pattern was equal to "BOWDOWN"
            if(nMatch == 100)
            {
                // turn to face the PC the NPC was talking to
                ActionDoCommand (SetFacingPoint
                (GetPosition(oWorship)));
                // and then play the worship animation for 30
                seconds
                ActionPlayAnimation (ANIMATION_LOOPING_WORSHIP,
                0.5, 30.0);
            }
        }
    }
}

```

This obviously isn't everything there is to know about factions and shouts, but hopefully the information is of use to some.

[Edited By David Gaider: Friday, 12 July 11:06PM (GMT)]

Celowin's Scripting Tutorial Lesson V - NonNPC Scripts

Introduction

The purpose of this sequence of lessons is to take a complete beginner to programming, and teach him or her how to use NWScript to write modules. The early lessons will be very basic, and anyone that has done any coding at all will be able to skip over them. The goal here is to make the lessons so that even the people that just shudder at any type of code can learn.

Feel free to post these lessons on any forum, print them out, or modify them. However, just give me credit for doing them.

Any comments on these lessons, good or bad, can be sent to me at james.foxglove@verizon.net

I am going to assume that anyone looking at these lessons has at least played around with the Aurora Toolset a bit. If there is enough feedback that people don't know how to do the simple placements that I have in these lessons, I will consider spelling out in more detail what needs to be done.

Other Resources

I've been getting a lot of requests for help on scripts lately. If I have time I'm glad to help point people in the right direction, but I don't always have the luxury of answering everyone that asks. So, I'm going to take a little bit of time to try and explain where else you can look to figure out what you need. Honestly, over 90% of the questions I see have been asked before, and it is only a matter of finding where the answers have been placed.

Really, there are four main places that you should look for answers, in no particular order:

- The Scripting FAQ (sticky thread in the scripting forum).
- The NWN Scripting Forum.
- The official campaign.
- The toolset itself.

Let's take a brief look at each of these, and how to use them.

The Scripting FAQ

Well, they are called "frequently asked questions" for a reason. The documents that have been compiled here answer many of the most confusing issues in scripting, and many

easier issues that are common for people to want to know about. Really, if you have an issue on something, it is a good idea to look here first. There may not be something that specifically deals with what you need to know... but then, there might be.

Also, don't be afraid to look at a document in the FAQ that doesn't immediately seem applicable to your problem. Odds are, everything in there is something you'll want to do at one time or another. The more you learn about NWScript, the more it starts to make sense. Often, while reading about how to do something totally unrelated to a problem I'm working on, I see something that can help me. It might be just a function that I wasn't aware of, or it might be a clever use of a variable that inspires me.

Because of the variety of different authors of the documents in the FAQ, some will be a bit beyond your current level of understanding. If so, fine.... learn more, and come back to them later. You might be surprised at how much you can pick up on even now, though. Give it a try.

The NWN Scripting Forum

There is a constant stream of people asking questions, and there are many dedicated people answering questions. As I was saying, odds are that any question you are likely to have has been answered already.

How then to find the answer? Well, the search tool is probably a good place to start. It isn't perfectly reliable, but it can't hurt to try.

If I don't find what I'm looking for through the search, then I just start scanning the first few pages of posts. Since the same questions are repeated so frequently, there is actually a good chance that what you are looking for has been answered within the first three pages.

If neither of those works, then, and only then, make a post asking for help. A quick note: the more effort you put out to solve your problem on your own, the more effort people are going to put out to help you. Let me give an example. (Any similarities to actual posts made by people is intended in general, but not meant to make fun of any individuals.)

One person posts, and the entire content of the post is "How do u make a lever close a door?"

Another person posts, and gives a detailed explanation of what they are trying to accomplish. They describe the tags they have given to their door and to the lever. They explain that they have tried attaching a script to the "OnActivate" of the lever, explain the local variables they have set, and explain what they have attempted to do in order to get the door to close.

I can't speak for everyone, but I'm much more inclined to help out the second person. Even putting aside my bias against the use of "u" as a pronoun, I don't feel any particular

sympathy for the first. Maybe that person has spent 15 hours trying to get it to work... but from what he has said, I don't know that. It comes across as "I'm not willing to put any effort into learning it, you do it for me." The second person, on the other hand, I can see is honestly giving it a try. It is easy to identify exactly where the problem is, and because of all the detail, I might even be able to write the code out for the person.

One final note about getting help from the forum: it is polite, though not strictly necessary, to give credit in your comments to the person who helped you fix your script. Just something like:

```
// On user defined script: tm_guard_ud
// Has a guard growl a warning when it sees a pc wielding a weapon.
//
// Last updated: 7/11/02
// Written by The Great Gatsby
// Some debugging help from Celowin of the NWN Scripting Forum
```

People who just play through your module will likely never see such credit, but those people that pick through your code will.

If you don't know a person's real name, the user name you know them by is fine. It is a good idea, though, to put in a blurb as I did above about where that user name comes from.

The Official Campaign

Over and over again, I see questions like "How do I do (mumble) like they did in (mumble) part of the official campaign?" I have to hope that this type of question is from a lack of knowledge of how to look at the official campaign modules, rather than laziness.

So, here are the steps to making the campaign modules available for opening in the toolset:

- Inside your main Neverwinter Nights directory, there is a subdirectory called nwm. This is where the official campaign modules are stored.
- Copy the file or files you want to look at from there into the modules folder.
- Rename the files in the modules folder from .nwm to .mod (When you do this, you won't see the .mod in the final result. That is, when you take Chapter1.nwm and rename it to Chapter1.mod, you will only see it show up as Chapter1)
- Right click on the renamed file, and go to "Properties." Uncheck the "Read Only" box, and then click ok.
- Now, whatever chapters you copied over can be opened in the toolset.

You can learn *tons* from seeing how BioWare did different things. Want to know how to have an npc initiate a conversation? Look at Pavel from the Prelude. Want to know how to summon in creatures to attack the pcs? Look at Aribeth's conversation from the Prelude. Want to know how to have an npc follow you until it sees a particular npc, then head off? Look at the butler or maid from Chapter 1. And so on.... they pulled all sorts of tricks in the campaign, all of them are now available for your perusal.

I personally go to the prelude first if I'm trying to figure something out... for the simple reason that it is faster to load up into the toolset. Certainly, there are lots of tricks used throughout the campaign that were not used in the prelude... but there is a lot of useful information even in that first little introduction to the game.

Sometimes their code can be difficult to follow, particularly for complicated things. However, if you can't figure out how to modify their code, odds are it is a bit above your current level of scripting. Work on something else, and come back to it.

Which actually, brings to mind something I've been meaning to mention for awhile. Usually, when people are starting work on their modules, they right away want to jump to the "cool part" which unfortunately is almost always the most complicated script. You have to learn to walk before you can run, Grasshopper. Not every script in your module is going to be complicated, start out with something easy, *even if it seems boring*. Work your way up to the more difficult ones. I remember reading awhile ago about someone who wanted his first script to involve six levers, twelve gems, a portal with multiple destinations, and lots of other visual effects. What he was doing was certainly possible, and not even all that difficult... once you know what you're doing. For a first script, though, it would be a logical nightmare.

The Toolset

It may seem a bit strange to think of the toolset as a way to learn about scripting. Honestly, though, this is where I've learned most of what I know. Once you know the basics of scripting, you can pull a *lot* of information out of the documentation in the toolset, minimal as it may seem.

First off, notice when you are editing a script, there is a long list of functions over on the right side. If you click on any one of those commands, you get a bit of information about that command down at the bottom of the screen.

Since most of the functions are pretty logically named, you often just scan the list until you find something that looks appropriate, and then look at it to see how it works.

Let's take an example from the script we're going to be looking at later in the lesson. I want to teleport a pc to a different location. I don't know off the top of my head what function to call to do that, so I start scanning through the list of functions.

We want to move the pc to a location, so ActionMoveToLocation sounds promising. However, we've used Move commands before, and it causes walking rather than a teleport. Right nearby in the list, though, is ActionJumpToLocation, and I have never seen my pcs doing any hopscotch in the game, so let's take a look at that. I click on it over on the right, and down below the following appears:

NWScript:

```
// The subject will jump to lLocation instantly (even between
areas).
// If lLocation is invalid, nothing will happen.
void ActionJumpToLocation(location lLocation)
```

The first two lines are comments, and tell us a little bit about the function we're looking at. The first line tells us that it is pretty much what we want, the second tells us something special about the behavior. What is the third line, then?

Well, that third line is telling us important information about how to use that function in the code. This really goes back to Lesson 1... the void tells us that there is no "output" from this function – it does something in game, but doesn't give us an answer of any kind. Then comes the name of the function, we knew that already. Then, however, it tells us what kind of *inputs* we need to pass to the function – we need to give it a location.

We can then hunt further through the function list to find other things that might help us. First off, this is an "Action" command, and so would make the teleport part of the action queue. If we want it to happen immediately, we need to find an alternative... what about just JumpToLocation? What do you know, it is a command.

Next problem is how do we get a location to pass to the function? Most of the functions that return an answer start with "Get" so let's look at that section. Yep, there is a GetLocation command. Again, we can click on it and get information on how to use it.

Hmmm... the JumpToLocation command doesn't take as an input what it is that you're trying to move. This one is a little bit tough to figure out, but I'll let you look for a bit. If you can't find it, it will be in the final scripts down below.

So, you see, with a little bit of perseverance, and maybe a bit of intuition (which comes with time) we can learn an awful lot about what commands to use.

Also, if you're bored sometime, I recommend just looking through the list sometime and clicking on random things that look interesting. Maybe you'll find something that you didn't know about that will make your life easier.

Moving On

Ok, enough about places to learn things, let's go ahead and make a script. Everything we've done in previous lessons has all dealt with npc scripting, so let's try something different. Let's do some placeable scripting, and some trigger scripting.

In general, no matter what you're scripting for, it is pretty much the same. The "triggers" for calling the script vary, and you have to be careful with what OBJECT_SELF refers to, but the basic structure and commands are exactly the same.

In our test module, if you've been following along with what I've been doing, we have two areas that aren't connected in any way. We've just been painting our module start point in whatever area as needed. While that is a great testing tool, it is boring. We could make an area transition from one place to the other... boring, again. So, let's try something a bit more interesting.

How about this: In one of the areas, we'll make two levers, that start in the "off" position. When both levers are moved to the "on" position, we'll summon a portal which a pc can step through to get to the other place.

Ok, step by step, here we go:

- Open the toolset, load up the module, and open the area you want to put your portal (presumably the larger of the two areas).
- Put down two "Floor Levers" from the "Placeable Objects", "Containers and Levers" menu
- Give one the tag LEVER1, the other LEVER2
- Where you want the portal to appear, put a waypoint. Tag it TM_INWP
- Paint a trigger around that waypoint (Click places for segments, after you have it pretty much enclosed, double click to close the polygon). Tag the Trigger PORTTRIG
- Attach this script to the OnUsed handle for each lever (same script for both levers). Save it as tm_lever_ou

NWScript:

```
// OnUsed script: tm_lever_ou
//
// This script sets up levers named LEVER1 and LEVER2.
// They both start deactivated. If both get turned on
// simultaneously,
// a portal is summoned at the waypoint TM_INWP and the trigger
// PORTTRIG is turned on.
//
// Written by Celowin
// Last Modified: 7/12/02
//
```

```

void main()
{
int nUsed1=GetLocalInt(OBJECT_SELF, "LEVER_STATE");
int nUsed2;
//
// nUsed1 and nUsed2 are used to temporarily hold the states of
the two levers.
// 0 is off, 1 is on
// LEVER_STATE is the permanent holding spot for the variables.
// Each lever stores its own LEVER_STATE
//
if (nUsed1 == 1) // If the lever is set to on, set it to off.
    SetLocalInt(OBJECT_SELF, "LEVER_STATE", 0);
else // If the lever is set to off, set it
to on
    SetLocalInt(OBJECT_SELF, "LEVER_STATE", 1);
nUsed1 = GetLocalInt(GetObjectByTag("LEVER1"), "LEVER_STATE");
nUsed2 = GetLocalInt(GetObjectByTag("LEVER2"), "LEVER_STATE");
if ((nUsed1==1) && (nUsed2==1)) // Are both levers on?
{
    // If so, create the portal, and tell the trigger to get
ready to port.
    object oPortalSpot=GetWaypointByTag("TM_INWP");

    CreateObject(OBJECT_TYPE_PLACEABLE,"plc_portal",GetLocation(oPort
alSpot), TRUE);
    SetLocalInt(GetObjectByTag("PORTTRIG"), "READY", 1);
}
}
}

```

There are things to discuss about this script, but for the most part I hope the comments make it clear. I'll hold off on explaining everything until the whole shebang has been set up.

- Go now to the trigger that you painted.
- Open up the properties, and go to the scripts tab there.
- Put this script into the "OnEnter" tab. Save it as tm_portal_en

NWScript:

```

// OnEnter script: tm_portal_en
//
// If the portal has been turned on, and a pc enters, warp that
pc to
// the waypoint TM_OUTWP
//
// Written by Celowin
// Last Updated: 7/12/02

```

```

//
void main()
{
// Set up the temporary variables that we need.
//
object oPC=GetEnteringObject();
object oDest=GetWaypointByTag("TM_OUTWP");
int nReady=GetLocalInt(OBJECT_SELF, "READY");
//
// Check: Is it a pc and is the portal turned on?
// If so, cause the pc to jump to the exit.
if ((nReady==1) && (GetIsPC(oPC)))
    AssignCommand(oPC, JumpToLocation(GetLocation(oDest)));
}

```

- Almost done. Ok out of the trigger.
- Now, switch to the other area, where you want to be teleported to.
- Paint down a waypoint, and call it TM_OUTWP
- Save everything, and go out of the editor.
- Load it up, and test it out.

The portal script is really straightforward. The only thing worth mentioning is the “AssignCommand”. This is another one of those things that once you find, you wonder how you ever got by without it. Basically, it is used to make *something else* do an action. We want the pc to do the JumpToLocation... so we assign the command to the pc. This comes in handy all over the place.

The other script isn’t really that hard to parse through either. The key is just the storing of the local variables to keep track of “on” and “off”. There are a few things that I think are worth mentioning, though.

It is sort of bad form for me to use nUsed1 twice in the script. Once I use it as the state of the current lever, whichever one it is. Later, I used it to mean the state of LEVER1 in particular. Some programmers would whack me with a large stick and say that I should use different variables for these. Why didn’t I? Well, just so I can raspberry the programmers that would try to tell me what to do. Seriously, the two uses are close enough to the same thing, and it is a simple enough script, that I felt there was no point in putting in another temporary variable. For a longer script, I probably would.

The other thing worth mentioning is the CreateObject call. For this, we need to understand the difference between “tags” and “blueprintrefs”.

Objects that are in the game have tags. You can find already created things through their tags. However, if something *hasn’t been made yet* it can’t have a tag. Imagine a physical

nametag... you can't put a nametag on something that doesn't exist yet.

Instead, we have to use a "blueprint", a plan for making the thing. That is what the "plc_portal" is... it is the blueprint reference for the portal object. (As a note, to find that blueprint reference, I just temporarily painted a portal somewhere. I then looked under the properties of that portal to find the blueprint.)

Other than that, I hope things are pretty clear. I tried to comment pretty extensively, so you can follow what I did.

Further Exercises

Ok, here is where my background as a teacher comes out.... homework time! I can't force you to do this, but I think it is worthwhile for those of you trying to learn this stuff to maybe take the time to extend what we've done.

So, try to do this: Right now, turning on both levers activates the portal. Fair enough. If we turn one lever on then off, then turn the other lever on, no portal. Still reasonable. However, if we activate both levers so the portal comes on, and then turn off one of the levers, the portal still stays up. This doesn't make as much sense. If you can, try to go through and fix it so that the portal turns off again when you turn off either one of the switches. Of course, it should then be possible to turn it back on again.

Another extension: Right now, our portal is "one way." We go in one place, and come out another. Try to make it two way.

I'd rate the first one as moderately challenging for someone that has followed the lessons so far, the second one as easy.

At any rate, good luck, and I hope to have another lesson out soon.

Celowin's Scripting Tutorial Lesson VI - Loops

Introduction

The purpose of this sequence of lessons is to take a complete beginner to programming, and teach him or her how to use NWScript to write modules. The early lessons will be very basic, and anyone that has done any coding at all will be able to skip over them. The goal here is to make the lessons so that even the people that just shudder at any type of code can learn.

Feel free to post these lessons on any forum, print them out, or modify them. However, just give me credit for doing them.

Any comments on these lessons, good or bad, can be sent to me at james.foxglove@verizon.net

I am going to assume that anyone looking at these lessons has at least played around with the Aurora Toolset a bit. If there is enough feedback that people don't know how to do the simple placements that I have in these lessons, I will consider spelling out in more detail what needs to be done.

For those of you that missed earlier lessons and are interested in looking them up, they have all been placed into the forum Scripting FAQ:
[Scripting FAQ](#)

A Word of Caution

It is with a bit of trepidation that I write and post this installment in my lesson sequence. Everything up to this point I consider to be "basic scripting." The techniques used in the past five lessons are going to end up being used in almost every script that you write. In this lesson, though, I'm going to talk about an intermediate concept. It is still a very important idea, it is just that you won't be using it nearly as often as the previous material.

That isn't the reason why I'm a bit hesitant though... the real reason why I worry is that this is the first lesson where what I am discussing is "dangerous." Certainly, it has been easy to make mistakes with some of the things we have talked about. If there has been a problem with a script, though, it is just that the script wouldn't work as intended. With loops, though, there is the potential to really screw things up.

Luckily, BioWare has some controls put in to keep things from getting **too** far out of hand. Still, you always need to be very careful to watch out for some of the problems that I'll be discussing.

What is a Loop?

In simplest terms, a loop is a way of getting part of your script to run multiple times. It goes through it once, then "checks" to see if it is supposed to run again. If so, it "loops" through and repeats itself. It does this over and over again, until things have changed to

the point where it isn't supposed to happen any more.

The key in writing loops is then mainly to know when to use them, and how to set up the "check" to make certain the loop stops at the right time. The check is particularly important – this is the danger I alluded to earlier. If your check is written wrong, there is the potential to never "break out" of the loop, and the same instructions will happen over and over until an error occurs.

There are two types of loops: the "for loop" and the "while loop." The first question that should pop into your head is "How do I know when to use each type?" I wish there were an easy, 100% accurate rule that I could give. The fact is, though, that knowing what type of loop to use is something that comes with experience. In fact, everything that can be done with a for loop could also be done with a while loop, which compounds the confusion – why have two types, when one would suffice?

There is, however, a general rule of thumb that we can use. The rule isn't perfect, but it covers most uses. If you are going to do something a certain number of times, you want to use a for loop. If you don't have any way of knowing how many times you are going to do something, you'll use a while loop.

Let's take a look at each type of loop separately.

For Loops

The key to a for loop is that you have an "index" variable that you use to keep track of going through the loops.

The basic format is something like this:

```
int nIndex;  
for (nIndex=1; nIndex <= 7; nIndex++)  
{  
do_things_here;  
}
```

This is a little bit complicated, let's look at each part specifically.

First off, we have to define the index variable first. (C coders are used to defining the variable inside the loop definition, but for NWScript it has to be declared first.) We don't have to give it a starting value, but we have to set up the variable.

There are three things inside the parentheses. The first one (`nIndex=1`) is the *initialization clause* we've set up the variable already, we just give it a starting value here.

The second part (`nIndex <= 7`) is called the *conditional*. This is the part that determines when the loop actually "runs." As long as this condition is met, the loop continues to run.

It is checked each time the loop tries to start again, and as soon as it is no longer true, the loop "terminates." So if after some loop, nIndex had a value of 8, the loop would stop running.

The final part (nIndex++) is an action that the code does at the end of each loop. I briefly mentioned the ++ part in a previous lesson... it increments the variable nIndex by 1. We could just as easily put in this third slot something like nIndex=nIndex+1. However, because incrementing by 1 is by far the most common "third clause" to the for statement, most experienced scripters prefer to use the ++. It takes a bit of getting used to, but it really does make the code look cleaner.

Putting all this together, what happens with the loop up there? nIndex starts out as 1. 1 is less than or equal to 7, so it performs the do_things_here stuff. Then the first iteration of the loop is over, so that third clause kicks in, and nIndex is incremented to 2.

It then goes back to the conditional... 2 is still less than or equal to 7, so the do_things_here runs again. The second iteration being over, nIndex is bumped up to 3. It goes through this a few more times, nIndex going through 4, 5, 6, 7 and then ending up as 8. At this point, when it checks, nIndex is now greater than 7, so the loop terminates.

Example For Loop

All this is pretty confusing stuff, so let's take a look at a concrete example. Let's say we want to punish a character for hacking open chests at random. So, we'll put in a chest that will summon 5 zombies and 5 skeletons around the character when it is destroyed. We could call the CreateObject command 10 different times, but instead it makes a lot more sense to use a loop.

(Note: I'm assuming that you have a pc available for testing that can survive that many zombies and skeletons. If you don't, you may want to change it so it summons chickens and commoners instead.)

- Place a chest into your module.
- On the "Lock" tab, set it to locked, and the difficulty to 100.
- On the scripts tab, the "OnDeath" handle, put the following script:

NWScript:

```
// On Death Script:  tm_chest_dt
//
// This script summons 5 zombies and 5 skeletons
// near the person that destroyed the chest.
//
// Written by Celowin
```

```

// Last Updated: 7/13/02
//
void main()
{
// Initialization: Get the location of the pc that destroyed the
chest,
// Set up for the loop.
//
object oPC=GetLastKiller();
int nUndeadIndex;
location lSpawn=GetLocation(oPC);
//
// Loop 5 times
//
for (nUndeadIndex=1; nUndeadIndex<=5; nUndeadIndex++)
{
// Each loop, create a zombie and a skeleton at the PC's
location
//
CreateObject(OBJECT_TYPE_CREATURE, "nw_zombie01", lSpawn,
TRUE);
CreateObject(OBJECT_TYPE_CREATURE, "nw_skeleton", lSpawn,
TRUE);
}
}

```

Go ahead and test it out. Destroy the chest, and suddenly your character is surrounded by an undead horde. Easy enough?.

Another thing to note is that in any of the three parts of the for loop initialization, we aren't restricted to just the one variable. Usually the only place you'll put something else in is the second one, but if desired you could alter the first or third as well.

Let's just modify that last script a tiny bit. Instead of summoning 5 each of skeletons and zombies, let's only summon zombies... but we'll summon a number equal to the level of the pc.

I won't write out the entire script, since most of it is the same. Here is what we need to change:

Under the line that starts "object oPC", add the line:

```

int nPCLevel = GetLevelByPosition(1,oPC) +
GetLevelByPosition(2,oPC) + GetLevelByPosition(3,oPC);

```

(You may want to break that up onto separate lines. As long as you only have the semicolon at the end, you're fine.)

Then, in the loop setup, the second part becomes `nUndeadIndex <= nPCLevel;`

Comment out the summon skeleton line, update the comments if you wish, and you're set to go.

Test it out, and you should be set. Check it out with a couple of different characters, and see how the number of zombies changes.

Wrap up on For Loops

Really, then, the key to using for loops is understanding the three parts. The initialization is run once, at the start of the process. The conditional is checked at the start of each loop, if it is true then the loop cycles, if not then the loop terminates. The third part is done at the end of each loop.

I can't stress enough that you have to be certain that your loop will end! What would happen if we had set our conditional to `nUndeadIndex > 0`? No matter how many times would increment the variable, this would always be true... so the script would summon zombie after zombie until an error occurred. It might be fun to see how many zombies it could make, but errors should be avoided at all costs.

So, always, always double check your logic when you're making a loop. Make certain that your loop will in fact terminate at some point.

On to While Loops

While loops are in many ways simpler to understand. However, I think they are harder to use, since there is less certainty about them. I mentioned before that you usually use a while loop when you don't know how many times you'll have to run the loop. That is a little bit confusing, so let me try to explain.

Suppose you have a bunch of hellhounds running around in an area. You want it so that when a certain altar is destroyed, all the hellhounds will suddenly die. The problem is, you can't be certain how many hellhounds there are.... The pcs may have killed a few. For that matter, maybe the altar will summon hellhounds until it is destroyed, so there could be lots around.

So, we really have no idea how many times we'll have to do a "kill hellhound" command. It might be none, it might be 50. There is just no way to tell up front.

That is where the while loop comes in. Basically, we want to say "keep killing hellhounds until none are left", or in other words "while there is a hellhound still around, keep killing them one after the other."

The basic format for a while loop is like this:

```
while (condition)
```

```
{
do_this_stuff;
}
```

It is really fairly straightforward. When the script gets to the while loop, it checks the condition. If it is true, it does the inside stuff, then checks the condition again. If it is still true, it repeats. It keeps doing this until the condition isn't true.

Simple enough, right? Really, in a lot of ways, it is like the for loop. In fact, you really have the same three parts... it is just that the initialization and the update you have to be sure to put in yourself.

We'll get to the hellhounds in a minute, first let's attack something a little bit simpler. It is a common thing when completing a quest to give the pcs a little bit of experience for their trouble. If you only have one pc, it is a piece of cake. But what if you are running a multiplayer game? How do you make sure each pc gets their reward?

If you didn't answer "with a while loop" give yourself twenty whacks with a wet noodle.

We have a few things floating around our module, if you've been following the lessons all along. (If you haven't, just make sure you have a ring with tag PASSRING somewhere, and any npc.)

Pick one of the npcs we have floating around, and we'll give it a simple little conversation: (I'm going to assume you know how to use the conversation editor to do this.)

```
(Start) Got ring?
(PC Response 1) No.
- End dialogue.
(PC Response 2) Yes.
- (NPC Response) Thanks.
```

(I'd actually make these a bit more verbose than this. I'll leave that to your own personal writing skills.)

For PC Response 2, just use the Script Wizard on the "Appears When" tab. Say that you want it to happen when "Item is in Inventory" and then add the tag PASSRING to the list.

For the NPC Response, on the "Actions Taken" tab, put this script:

```
NWScript:
// Conversation Script: tm_guard_c1
// Takes the ring from the speaking player,
// And awards 50 xp to every pc.
//
```

```

void main()
{
object oPC=GetFirstPC();
DestroyObject(GetObjectByTag("PASSRING"));
//
while (oPC != OBJECT_INVALID)
{
    GiveXPToCreature(oPC, 50);
    oPC=GetNextPC();
}
}

```

Save the script, save the conversation, save the module, then go test it out. You'll have to have a friend help you out to really test it properly, but you can see the basics just by running it yourself.

Notice that ignoring the DestroyObject command, really what we have it just the basic steps of the for loop. We initialize (oPC = GetFirstPC()), we condition (oPC != OBJECT_INVALID) and we update our variable (oPC = GetNextPC()).

So, this will keep looping, giving 50 xp to each pc, until we come up with OBJECT_INVALID, that is, until there are no more pcs.

Once again, when writing a while loop, be absolutely certain that the loop will eventually terminate. Infinite loops are bad.

Exercises

Hmmmm.... I was going to go through and do the hellhound script, but the more I think about it, the less I think I have to. In fact, basically every part of the script has been used somewhere in this lesson.... What handle to use, what commands go in the script, everything. So, I'll leave that to you... drop an altar and some hellhounds into your module, and see if you can make it so that destroying the altar will destroy all the hellhounds. (Difficulty = moderate. You can also make it so that the altar will spawn hellhounds until it is destroyed.... Difficulty remains moderate if you just keep spawning them, difficulty ups to advanced if you make sure that the number of hellhounds never gets above a certain level.)

Another exercise to try: Make a trigger, that when entered will summon a number of creatures (whatever kind you like) equal to twice the number of pcs in the module. (I'll rate this one as advanced, but barely so.)

Closing

I've been getting less and less feedback on these tutorials as I've been going on? Are people still finding them useful? Please, *please* let me know if you're still learning from them. I don't get paid for this, the only satisfaction I get is from knowing that I'm helping

others to make cool modules.

In fact, probably the best praise that you could give me is to show me what you've learned. Use the ideas I've covered, write something cool, and tell me about what you did. After lesson four, someone showed me an extension of the dart player script that just floored me. I really felt good about myself that I had helped that person get started.

At any rate, just drop me a line to let me know how well things are going. If nothing else, let me know if you've been able to figure out the exercises I've been dropping in.

As a reminder, you can always contact me at james.foxglove@verizon.net

Celowin's Scripting Tutorial Lesson VII – Scripting for Items

Introduction

The purpose of this sequence of lessons is to take a complete beginner to programming, and teach him or her how to use NWScript to write modules. The early lessons will be very basic, and anyone that has done any coding at all will be able to skip over them. The goal here is to make the lessons so that even the people that just shudder at any type of code can learn.

Feel free to post these lessons on any forum, print them out, or modify them. However, just give me credit for doing them.

Any comments on these lessons, good or bad, can be sent to me at james.foxglove@verizon.net

I am going to assume that anyone looking at these lessons has at least played around with the Aurora Toolset a bit. If there is enough feedback that people don't know how to do the simple placements that I have in these lessons, I will consider spelling out in more detail what needs to be done.

For those of you that missed earlier lessons and are interested in looking them up, they have all been placed into the forum Scripting FAQ: [Scripting FAQ](#)

Introduction

I've been seeing a lot of questions like "How do I attach a script to an item?"

Well, the short answer is "you can't." That isn't a very satisfying response, though, so let me explain further. For whatever reason (my guess has to do with balance on exported characters) BioWare has not allowed any way to directly put a script to an item. However, you can still script actions for an item, but you have to do it in a manner that seems somewhat roundabout.

When editing an item, under the "cast spell" option, there are two extra choices: Unique Power, and Unique Power Self Only. By using these two things, we can add all sorts of special abilities, if we go about it right. The "Unique Power Self Only" option is for items that are activated and go off, the plain "Unique Power" option is for abilities that need to be targeted.

If we can't attach any scripts to the items, though, how can we make them do anything? Well, go to Edit -> Module Properties -> Events tab, and you will see a place for attaching scripts that apply to the entire module. One of the options is "OnActivateItem", and this is where all the fun happens.

There are four functions that we will need to understand in order to script for items:

GetItemActivated – This returns the item that was used.

GetItemActivatedTarget – If the item was targeted on an object, this returns it. (Remember, of course, that creatures are objects.)

GetItemActivatedTargetLocation – If the item was targeted on the ground, this returns the location. If the item was targeted on an object, this returns the location of that object.

GetItemActivator – This returns the pc that activated the item.

These descriptions are very brief, but we'll be seeing each one in action through our examples.

Example 1: Flawed Healing Amulet

Let's make an amulet that can be used to heal the pc. To make it a bit more interesting, though, let's say that it isn't a perfect heal – it heals half of the damage that the pc has suffered, and to keep it from being too overpowered, we'll restrict it to one use per day.

First, create the base item:

- Open up the item wizard.
- Select "Amulet" as the type.
- Give it the name "Cracked Amulet"
- Check the "magical" box, Item Level 1-5, Low quality
- On the next screen, put it on the palette under Miscellaneous -> Jewelry -> Amulet
- Finally, check the "Launch Item Properties" box, and finish.
- On the General tab, change the tag to HEALNECK
- On the Properties tab, remove whatever abilities it currently has
- Add in the power Cast Spell -> Unique Power Self Only
- Select that ability on the right, then underneath click on "Edit Property"
- Click on "1 Use/Day", Select, then OK
- For ease of testing, check the Identified box.
- OK out of the item properties.

Now, for the scripting. Go to Edit -> Module Properties -> Events, and edit the OnActivateItem script. Put in the following script:

NWScript:

```
// Module OnActivateItem Script:  tm_activate
//
// This script handles all the unique item properties for the
module.
// Currently scripted items:
// Cracked Amulet
//
// Written By: Celowin
// Last Updated: 7/15/02
//
void main()
{
// General Initialization:  Find what item was used
object oUsed=GetItemActivated();
//
// The following section is for the Cracked Amulet
// The amulet heals half the pcs damage when used.
if (GetTag(oUsed)== "HEALNECK")
{
// Get the pc, first.
object oPC=GetItemActivator();
//
// Find out how much damage was taken, we're healing half of
that.
int nDamageTaken = GetMaxHitPoints(oPC)-
GetCurrentHitPoints(oPC);
int nHealing = FloatToInt(IntToFloat(nDamageTaken)/2);
//
// Perform the healing, and a little visual effect to show it
working.
effect eHeal = EffectHeal(nHealing);
ApplyEffectToObject(DURATION_TYPE_INSTANT, eHeal, oPC);
effect eVisual = EffectVisualeffect(VFX_FNF_SMOKE_PUFF);
ApplyEffectToObject(DURATION_TYPE_TEMPORARY, eVisual, oPC,
1.0);
}
}
```

Save the script, and go test it out. (Don't forget to drop a copy of the amulet into your module, if you haven't done so yet.)

There are two things about this particular script that bear mentioning. The first is the line:

```
int nHealing = FloatToInt ( IntToFloat( nDamageTaken ) / 2 );
```

Now, the basic idea here is pretty simple: we want to cut the damage taken in half in order to get how much we heal. What is all this IntToFloat and FloatToInt stuff, then? Well, remember that an int is a whole number. There is no decimal allowed there. When we divide two numbers, though, odds are we are going to get a decimal part. So, using an integer in a division is always suspect.

A "float" is a number with a decimal part. So, since we're doing a division, the first thing we want to do is convert the integer `nDamageTaken` into a float... hence the `IntToFloat`. Then, since we have a decimal number, the division is allowed.

But then, after the division occurs, we need our final result to be an integer... so we do the reverse, `FloatToInt`. This basically "chops off" the decimal part of the number, and leaves us with the whole number part.

Let's look at an example of the math: Say a pc has taken 17 points of damage. We change that to a float, so it becomes 17.0. Then we divide by 2, it becomes 8.5. We change it back to an integer, the decimal part is dropped, and we're left with 8. So, we would be healing 8 points of damage.

The second part of the script worth mentioning is all the "Effect" stuff down at the bottom. Pretty much any time you are going to directly affect an object in any way, you need to use effects. Healing, visuals, special abilities, knockdowns.... all these are effects of one kind or another.

I'm doing two effects here... one is the heal, and the other is a visual of smoke around the player. (Since the amulet is supposed to be not working quite properly, I thought the smoke was appropriate.) Each effect has two parts... defining the effect, and then applying it to the pc.

For the heal, we first have to tell it how much to heal. That is handled in the line:

```
effect eHeal = EffectHeal(nHealing);
```

Then, we heal the pc.

```
ApplyEffectToObject ( DURATION_TYPE_INSTANT , eHeal , oPC );
```

The `DURATION_TYPE_INSTANT` says that it happens and then is over. `eHeal` is the effect we defined, then `oPC` is the target of the effect.

The second one is similar. We define the visual effect of a smoke puff, then apply it to the pc. This one is going to run for a few seconds, so it is `_TEMPORARY` for the duration type. We again say what effect it is, and the target. The final number is the duration.

There are so many different possible effects that it is often really hard to find the one you want. All of them are applied in basically the same manner, though.

Example Two: Alfred's Ring

For those of you just interested in scripting, skip this paragraph. I'm going to get maudlin and remember one of my old friends who was a fantastic D&D player and DM. At one

point he was in humorous campaign that unfortunately I was not a part of, though I heard plenty of stories. His character was "Alfred the were-chicken." I don't remember the back-story of the character, but he had all the usual lycanthropic powers... shape shifting under a full moon, immunity to non-silver weapons, and of course, control over appropriate animals. Well, if anyone has a copy of the first edition AD&D players handbook, you can look up the cost of a chicken under livestock... 1 copper piece. There were 200 copper pieces to a gold. So, for 2 gold, he had an army of 400 chickens following him around. Sure, an individual chicken can't do much... but if you're mobbed by 400 of them at once, all clawing and pecking.... <shudder>. At any rate, this item is in remembrance of Alfred the were-chicken.

The way this ring will work is to summon a chicken. If we click on the ground, it will just create the chicken. If we click on a creature, the chicken will attack that creature. (Well, for a moment... chickens aren't known for their valor in battle.)

Go ahead and create the item. It is basically the same process as for the first item, so I won't go through it step by step. Give it a name of Alfred's Ring, a tag of CHCKRING, and unlimited uses per day. (If it were a useful creature, that would be unbalancing. But a chicken?)

Now, to make this item work, we have to add to the same script. If we still want the healing ring to work, we have to leave all the former stuff in. We just put in another section to check for the new item we're scripting for.

NWScript:

```
//
// The following section is for Alfred's Ring
// Summons a chicken, which will attack the target if a creature
if (GetTag(oUsed) == "CHCKRING")
{
    // Get where the ring was used
    location lTarget=GetItemActivatedTargetLocation();
    // Set up the "gate in" effect
    effect eChickIn=EffectVisualeffect(VFX_FNF_SUMMON_MONSTER_3);
    // Summon the chicken
    object
oNewChicken=CreateObject(OBJECT_TYPE_CREATURE,"nw_chicken",
lTarget, TRUE);
    // If the ring was targeted on a creature, the chicken will
be summoned offset a bit.
    // Update the location, then apply the effect.
    lTarget=GetLocation(oNewChicken);
    ApplyEffectAtLocation(DURATION_TYPE_TEMPORARY, eChickIn,
lTarget, 1.5);
    //
    // Finally, if check to see if an object was targeted. If
so, attack it.
    object oEnemy=GetItemActivatedTarget();
    if (GetIsObjectValid(oEnemy))
        AssignCommand(oNewChicken, ActionAttack(oEnemy));
}
```

}

Also, update your header comments. (I know, it is boring to update comments every time, but it is a really good practice to get into.)

Go ahead, save, and test it out. (Note when testing: the chicken won't actually attack anything that it is friendly with... so you might need to play with factions a bit to get it to work. Once you do, it is sort of funny... the chicken will attack, then get scared and run away.)

Some Cautionary Notes

Although I'm sure people are brimming with ideas of neat items you can script, you really should limit yourself to only a few in a module. One reason is stylistic: the more such items you put in, the less impact they have on a person.

Another reason is for efficiency... since all the effects get added to the one script, that script can get *huge* if you put in too many items.

The other warning about items is that you need to remember that they won't work in another module. You can certainly cut and paste your OnActivateItem script to other modules, but it won't happen automatically.

For these reasons, it is common to restrict unique item powers to items needed for the plot. You are not forced into it, but it is something that you may want to consider.

Exercises

I would imagine that people have their own ideas on how to use this, but if you want some practice here are a few things to try:

1. Make two altars in different places. Praying at one of these altars will set it to be your recall point. Then, using an item will teleport you to that altar. (Moderate)
2. If your pc goes overboard with Alfred's ring, too many chickens might burden your system. Modify it so that the chicken "unsummons" after a certain amount of time. (Easy if you find the right command.)
3. Create a charged wand that will heal the target, but take 100 xp away from the user. (Moderate, or Difficult if you put in a beam effect for the wand.)

Anyway, go have fun with it!

Celowin's Scripting Tutorial Lesson VIII - Functions

Functions

Introduction

The purpose of this sequence of lessons is to take a complete beginner to programming, and teach him or her how to use NWScript to write modules. The early lessons will be very basic, and anyone that has done any coding at all will be able to skip over them. The goal here is to make the lessons so that even the people that just shudder at any type of code can learn.

Feel free to post these lessons on any forum, print them out, or modify them. However, just give me credit for doing them.

Any comments on these lessons, good or bad, can be sent to me at james.foxglove@verizon.net

I am going to assume that anyone looking at these lessons has at least played around with the Aurora Toolset a bit. If there is enough feedback that people don't know how to do the simple placements that I have in these lessons, I will consider spelling out in more detail what needs to be done.

For those of you that missed earlier lessons and are interested in looking them up, they have all been placed into the forum Scripting FAQ: [Scripting FAQ](#)

Introduction

This lesson is going to be a difficult one, and I'm not even going to really be scratching the surface of the topic. Ever since lesson one, we've been using functions that have been written by BioWare. This time, we're going to start to learn how to write our own functions.

Don't worry if as this lesson goes on, you don't understand everything that I say. This stuff is never learned in one sitting, and in fact takes a lot of practice before it makes sense. If you feel you're getting over your head, put it aside for a bit, go do something else, and come back to it later.

A bit of warning before we start. Even though we are going to be writing functions, they will only be able to be used "locally." That is, imagine we are writing a script for a particular handle, say the OnPerception for an npc. We write a function to use in that script. We can use it in our "main" for that script, but we can't use it anywhere else. We couldn't use it for any other handle on that npc, nor could we use it for a different OnPerception script.

Yes, this limits us a lot as to what we can do with the functions we write. At the same time, it still opens up a lot of possibilities. And fear not, in a future lesson I will explain

how to write more generically useful functions, but we have to take things one step at a time.

Function Declaration

Every function will start with a line something like this:

```
void GateIn(string sBluePrint, location lGate)
```

It is a brief line, but there are a lot of complicated concepts tied up into it.

The first part, “void” tells us what the output for the function will be. So, in this case, the output is going to be an integer. All the other data types we have been using could go here: int, float, object, event, etc. The most commonly used one will be void, and for this lesson it will be the only one that we deal with.

The next part, “GateIn” names the function. Once we have it defined, we can call the function using this name, just like all the BioWare defined functions. (Though again, we must remember the restriction that we can only call our function in the script we are writing it for.)

The part inside the parentheses is the most difficult part to understand... here we are setting up the inputs to our function. (The technical term for these is “parameters,” but you probably don’t need to remember that.) We are saying that we are going to have two inputs into our function... one string, and one location.

So far, so good. Now for the confusing part. Before we can actually write the function, we need to understand what will happen when the function is called. (Right away, that seems backwards, that we need to understand the call of the function before we write it, but bear with me a moment.) Presumably, we are writing a function that will do something with those inputs, otherwise we wouldn’t need them.

Suppose, then, that somewhere in our script, we call the function like this:

```
GateIn(“nw_fireelder”, lSummonPoint);
```

Now, then, “nw_fireelder” is our first input, a string. Effectively, the first thing our function does is set sBluePrint equal to “nw_fireelder”. Anywhere in our function, we can use the variable sBluePrint to stand for this.

The same thing for our second input. lGate is set to whatever location lSummonPoint is holding.

Ok, let’s look at the whole function now.

NWScript:

```
void GateIn(string sBlueprint, location lGate)
// This function summons the creature with blueprintresref given
by sBlueprint
// at the location lGate
// Then, it has the summoned creature attack the closest pc to
the object that
// calls this function.
{
// Creates the creature
object oNewCreature=CreateObject(OBJECT_TYPE_CREATURE,
sBlueprint, lGate);
// Find the closest pc
object oPC=GetNearestCreature(CREATURE_TYPE_PLAYER_CHAR,
PLAYER_CHAR_IS_PC, OBJECT_SELF);
// Cause the creature to attack the pc
AssignCommand(oNewCreature, ActionAttack(oPC));
}
```

Note that if you type this up and try to compile it, you will get an error message:

ERROR: NO FUNCTION MAIN() IN SCRIPT

By itself, a function isn't a script. You still need your main, but for now I just want to look at the function by itself.

Probably the most important line is the first one, where we create the object. We are just using the same old CreateObject routine we've used many times before, but we are passing our inputs to it. Whatever inputs are given to GateIn when it is called, are then passed along to the CreateObject function.

It seems confusing, but it is here that the real power comes in. We can call the GateIn command multiple times, passing it different inputs, and it will summon the different creatures.

The next line, starting object oPC= is long, but mainly it is just because of the things we have to tell it. That whole line is just saying "find the nearest pc to OBJECT_SELF." We don't know what OBJECT_SELF is at the moment, because we don't know what object is actually calling the GateIn function.

Finally, then, we tell the new creature to attack the pc we just found. Nothing major there.

Let's Try it Out

I'm going to write a fairly complicated script using the function I just showed. We could certainly do something a lot easier than this, but I want to do an example where we can see *why* we'd want to use a function.

- Start up the toolset.
- First, use the item wizard to create a Miscellaneous Medium item.
- Give it the name Sorcerer's Skull
- Put it under Plot Items on the palette
- Finish the wizard, and edit the properties
- Give it the tag ALTSKULL
- Change the appearance to iit_midmisc_021
- OK out.
- Paint a waypoint, tag it ALTSUMWP
- Place an altar nearby, edit the properties.
- Tag the altar with SUMMALTR
- Check the "usable" and "has inventory" boxes.
- Open the altar's inventory, put in a copy of the skull we just made.
- Ok out of the inventory, go to the scripts.
- In the OnDisturbed handle, put the following script:

NWScript:

```
// OnDisturbedScript: tm_summaltr_ds
//
// This script gates in one of 10 random creatures
// when a skull ALTSKULL is removed from the altar.
// The creature is gated in at the waypoint
// ALTSUMWP
//
// Written by Celowin
// Last Updated: 7/16/02
//
void GateIn(string sBlueprint, location lGate)
// This function summons the creature with blueprintresref given
by sBlueprint
// at the location lGate
// Then, it has the summoned creature attack the closest pc to
the object that
// calls this function.
{
// Creates the creature
object oNewCreature=CreateObject(OBJECT_TYPE_CREATURE,
sBlueprint, lGate);
// Find the closest pc
object oPC=GetNearestCreature(CREATURE_TYPE_PLAYER_CHAR,
PLAYER_CHAR_IS_PC, OBJECT_SELF);
// Cause the creature to attack the pc
AssignCommand(oNewCreature, ActionAttack(oPC));
} // end function GateIn
//
// Here is our main function:
void main()
{
// If the skull is in the altar, we don't care, nothing will
```



```

happen.
// Note that OBJECT_SELF refers to the altar
if (GetItemPossessor(GetObjectByTag("ALTSKULL")) != OBJECT_SELF)
{
    // Find the summon spot, via the waypoint.
    location
lSummonPoint=GetLocation(GetWaypointByTag("ALTSUMWP"));
    // Create the visual effect for the gate.
    effect eGate=EffectVisualeffect(VFX_FNF_SUMMON_GATE);
    ApplyEffectAtLocation(DURATION_TYPE_TEMPORARY, eGate,
lSummonPoint, 3.0);
    // Randomize what creature is being summoned.
    int nCreature=d10();
    switch(nCreature)
    {
        case 1: // Summon a polar bear.
            DelayCommand(3.0,GateIn("nw_bearpolar", lSummonPoint));
            break;
        case 2: // Summon a cow.
            DelayCommand(3.0,GateIn("nw_cow", lSummonPoint));
            break;
        case 3: // Summon a bone golem.
            DelayCommand(3.0,GateIn("nw_golbone", lSummonPoint));
            break;
        case 4: // Summon an elder fire elemental.
            DelayCommand(3.0,GateIn("nw_fireelder", lSummonPoint));
            break;
        case 5: // Summon an ogre high mage
            DelayCommand(3.0,GateIn("nw_ogremageboss", lSummonPoint));
            break;
        case 6: // Summon a yuan ti mage
            DelayCommand(3.0,GateIn("nw_yuan_ti002", lSummonPoint));
            break;
        case 7: // Summon a spitting fire beetle
            DelayCommand(3.0,GateIn("nw_btlfire02", lSummonPoint));
            break;
        case 8: // Summon a Kreshar
            DelayCommand(3.0,GateIn("nw_kreshar", lSummonPoint));
            break;
        case 9: // Summon a werecat, human form
            DelayCommand(3.0,GateIn("nw_werecat001", lSummonPoint));
            break;
        case 10: // Summon a high lich
            DelayCommand(3.0,GateIn("nw_lichboss", lSummonPoint));
            break;
    } // end switch
} // end if
} // end main

```

Save everything, and go test it. When you remove the skull from the altar, a random one of the 10 creatures will gate in and attack (well, the cow won't attack, but the others will).

As long as it is, our main function really isn't all that complicated. First, we check to see who has the skull... if it is anything but the altar, we go forward.

We create a gate visual effect at the waypoint. This is just like the effects we did last lesson. Then, we get a random number from 1 to 10, and call our gate in function to summon the creature.

Now that we have this example in front of us, let's discuss why we wanted to use a function here. There are actually two reasons... one that is pretty straightforward to understand, and another that is a bit more complicated.

Let's discuss the easy one first. Basically, every time we call the GateIn function, we are saving ourselves from writing out three lines of script. By just calling the GateIn function, we are summoning the creature, finding the target, and attacking the pc all in one. Since we have 10 different cases, we've saved ourselves about 20 lines in our script, not counting comments. (And given how ugly that function was to find the pc, I'm glad not to have to have it in my script repeatedly.)

The second reason is a bit tougher to understand... but basically, we *had* to use a function in this case. We wanted to delay the summon until the gate was formed, so that the fire effect would mask the appearance of the creature. Thus the need for the DelayCommand. However, we can only DelayCommand things that have an output of "void", and the CreateObject command returns an object. If you try to do a DelayCommand for a CreateObject call, it won't compile. By putting the CreateObject into a separate function, that *did* return void, we get around that restriction.

Cleanup

I think this lesson is a bit too complicated to expect you to be writing your own functions just yet, but I'll have you fix the previous script a bit.

The way it is now: If the skull is in the altar, everything is peachy. You can add items to the altar and remove them, as long as the skull remains. As soon as you remove the skull, a creature is gated in. All this is fine.

However, what if you continue to play with the altar? If you hold onto the skull, and put any other item into the altar, it will gate in another creature! Remove that new item, and it gates in another! This may be what you want, but probably not.

So, fix it so that only when the skull is removed will the creature be summoned. You actually don't have to change the function at all, you just need to play around with the conditional for your main.

Example 2: Removing Plot Items

A certain sick, twisted DM (oh wait, that was me) once designed a surreal dream sequence for his pcs. Talking penguins, nonsequiturs, bizarre puzzles, the works. Eventually, though, the pcs woke up. <sniff> In pen and paper, it is easy to just get rid of

all the items the players picked up in the dream world. But how about in NWN?

Well, it takes a bit of planning, but it can be done. First off, I named all my dream world items with similar tags. All plot items were DREAMITM, all weapons were DREAMWPN, all armor was DREAMARM, and the key (there was only 1) was DREAMKEY.

Then, to the dream world OnExit handle, I attached this script:

NWScript:

```
// On Exit Area script:  tm_area002_ex
//
// This removes all items with tag DREAMITM, DREAMWPN,
// DREAMARM, or DREAMKEY from the exiting pc.
//
// Written by Celowin
// Last updated: 7/16/02
//
void StripItems(object oStrippee, string sTag)
// This function strips all items with tag
// sTag from the object oStrippee
{
// Initialize: Get the first inventory item
object oCurrentItem=GetFirstItemInInventory(oStrippee);
// Loop through all items in inventory
while (oCurrentItem != OBJECT_INVALID)
{
if (GetTag(oCurrentItem)==sTag)
DestroyObject(oCurrentItem); // Destroy items with correct
tag
oCurrentItem=GetNextItemInInventory(oStrippee);
} // end while
} // end StripItems function
//
//
void main()
{
// Start main script
// First, get the one exiting
object oPC=GetExitingObject();
if (GetIsPC(oPC)) // If it is a pc, strip the items
{
StripItems(oPC, "DREAMITM"); // Generic dream items
StripItems(oPC, "DREAMWPN"); // Dream weapons
StripItems(oPC, "DREAMARM"); // Dream armor
StripItems(oPC, "DREAMKEY"); // Dream key
} // end if
} // end main
```

(Note, there is actually a better way of doing this with some string manipulation... this version is really inefficient, since it loops through the pc inventory four times. It works,

though, and since it will only be run once per pc if you design the module right, it isn't a big deal.)

You can test this if you want. Create and drop a few DREAM*** items into your module, attach the script to one of the areas' on enter scripts, and play around with it. Also, for you cruel dms, this can easily be modified to remove every item from a player...

Wrap Up

I've just barely scratched the surface of functions here. We can write functions to calculate things for us, we can create libraries of functions, we can use functions to drastically reduce the complexity of some scripts. Because of the power of functions, though, it is difficult to cover everything at once.

For the most part, the feedback I've gotten from people has been positive. There is only one minor complaint that comes up repeatedly, something like this: "I can follow what you do, and you explain yourself well. So, I can see the how and the why. But I have a tough time figuring out the *when*. I never know when I should use one of the tools you have shown, and when I should do something else."

I try to explain that, as well, but the problem is that it is something that comes with time. After you've looked at enough scripts, you start to develop an intuition about when to use a certain technique. This isn't to say that it is easy... even once I know what I'm doing, I often bang my head against the keyboard for hours to get a complex script to work.

So, for functions, what is the "general rule" on when to use them? I'd say, look at what you're doing. If you find yourself writing the same things over and over, odds are that you want to use a function to simplify it. Even if it is only a few lines that you find yourself repeating, your code will end up much easier to understand if you write a function to encapsulate those lines.

The Future

I'm really starting to run out of ideas for these lessons. I've really covered all the "basics," and I'm actually starting to touch on some pretty advanced concepts. Sure, there are little details here and there that I've glossed over, things that I've omitted, but I would think at this point that you should be able to look at almost any script and say "Hey, I know what that does." Maybe you couldn't write it yourself from scratch, but you should be able to dissect it and figure out what it does.

I certainly should talk a little bit about "include" statements, and writing your own function libraries, but I don't think that will be a very complicated lesson.

So, unless I'm struck with a sudden inspiration, I think this series will be ending around the 10th tutorial – at least in this form. I have an idea for a "follow-up" series, that will take a different approach. I'll give more details on that after I hammer out a few details.

Celowin's Scripting Tutorial Lesson IX - More on Functions: Functions that Return, Default Parameters, Libraries

Introduction

The purpose of this sequence of lessons is to take a complete beginner to programming, and teach him or her how to use NWScript to write modules. The early lessons will be very basic, and anyone that has done any coding at all will be able to skip over them. The goal here is to make the lessons so that even the people that just shudder at any type of code can learn.

Feel free to post these lessons on any forum, print them out, or modify them. However, just give me credit for doing them.

Any comments on these lessons, good or bad, can be sent to me at james.foxglove@verizon.net

I am going to assume that anyone looking at these lessons has at least played around with the Aurora Toolset a bit. If there is enough feedback that people don't know how to do the simple placements that I have in these lessons, I will consider spelling out in more detail what needs to be done.

For those of you that missed earlier lessons and are interested in looking them up, they have all been placed into the forum Scripting FAQ: [Scripting FAQ](#)

Introduction

It has been awhile since my last lesson, and this one is going to build heavily on that one. So if you were at all shaky on lesson 8, I recommend re-reading it. You don't have to have it mastered, and in fact I would expect that most people won't be that far along. In fact, I'm hoping that seeing more examples in this lesson will help people that are still a bit confused on the subject. However, if you don't at least get the general idea of the last lesson, this one won't make much sense at all.

We're going to be taking the idea of the function from the last lesson, and extending that idea in a couple of different ways. The last time, we talked about the very basics of using a function, and while there were uses for it, the concept was very limited. Here, we're going to add a few things to make functions much more versatile.

Functions that Return

Up until this point, every function that we've written ourselves has been an "action." That is, it has done things, but it hasn't returned an answer. We specified that our function was an action by declaring it as type "void."

My guess is that nearly all of the functions you will end up writing will end up being actions. However, there are a few times when you need to *calculate* something. Let's do a simple example... suppose we have a mage's guild. As they are fond of their privacy, their tower can only be accessed through a magical portal, and that portal makes sure only to transport in people whose primary focus is magery.

The way I'm going to handle this is to write a function to check the percentage, by level, of a character's magical classes. A pure level 1 wizard with no multiclass would return 100%. A second level fighter, third level sorcerer multi would return 60%. Both of those would be let in, but the level 7 rogue, level 2 wizard would only be 22.22%, and would be left out in the cold.

First, let's write our function. We want it to take in an object (a creature), and spit out the percentage. We'll give the percentage in decimal form. If you remember, a decimal number is type "float." So to declare the function, it is:

```
float GetMagePercent(object oPerson)
```

Really, the calculation done here is fairly easy. We need to get the total level of the creature passed to the function. (I did this before with `GetLevelByPosition`, but I've since realized that `GetHitDice` is a much simpler solution.) We also need the levels of wizard and sorcerer, easily gotten with `GetLevelByClass`. A simple division, and we have our percent.

The thing is, how do we tell the function we are writing that the number we found is our "answer"? It is fairly simple... we tell it to "return" the value. Let's take a look at the entire code.

NWScript:

```
// This function will return the percent of a creature's
// levels that are mage-related, expressed as a decimal
// number. It will return 0.0 if the object passed to it
// is not a creature.
float GetMagePercent(object oPerson)
{
    int nTotalLevel=GetHitDice(oPerson);
    if (nTotalLevel==0) // if not a creature, return 0.0
        return 0.0;
    else // else, calculate the percent and return it
    {
        int nMageLevel=GetLevelByClass(CLASS_TYPE_WIZARD,
oPerson)+GetLevelByClass(CLASS_TYPE_SORCERER, oPerson);
        float
fMagePercent=IntToFloat(nMageLevel)/IntToFloat(nTotalLevel);
        return fMagePercent;
    }
}
```

Most of it should be familiar to you, it is just the last line that is new. We've spent the rest of the script calculating fMagePercent, the last line just says that when the function is called, that will be the output.

A further note on what I did: Notice that I am careful about what happens if GetHitDice returns 0. If we use our function properly, that should never happen. However, the more "checks" you put in to deal with errors, the better. It may be that a weapon accidentally gets passed to our function. If that were to happen, we'd get a "division by 0" error, and cause all sorts of havoc.

Now that the function is written, how do we make use of it? Well, pretty much like any BioWare written function. (Again, remember that for the moment, we have to put the whole function code into our script where we want to call it.)

So, plop down a portal placeable (I actually used a magic sparks, for variety), and put a trigger around it. Paint a waypoint elsewhere, and call it WIZTOWER_WP

Put the following into the OnEnter code for the trigger:

NWScript:

```
// This function will return the percent of a creature's
// levels that are mage-related, expressed as a decimal
// number. It will return 0.0 if the object passed to it
// is not a creature.
float GetMagePercent(object oPerson)
{
int nTotalLevel=GetHitDice(oPerson);
if (nTotalLevel==0) // if not a creature, return 0.0
return 0.0;
else // else, calculate the percent and return it
{
int nMageLevel=GetLevelByClass(CLASS_TYPE_WIZARD,
oPerson)+GetLevelByClass(CLASS_TYPE_SORCERER, oPerson);
float
fMagePercent=IntToFloat(nMageLevel)/IntToFloat(nTotalLevel);
return fMagePercent;
}
}
void main()
{
object oPC=GetEnteringObject();
if (GetIsPC(oPC) && (GetMagePercent(oPC) > 0.50))
AssignCommand(oPC,
JumpToLocation(GetLocation(GetWaypointByTag("WIZTOWER_WP"))));
}
```

It might not be a bad idea to break the AssignCommand line into several steps... first get the waypoint, then the location, and then jump to that location. It is sort of a matter of style... here, I can follow the levels of nesting fairly easily, so I don't feel the need to

break it down into steps.

Another Example

Here's another example of a function to calculate something. Sometimes (well, ok, often), I like to be a little bit sadistic to my pcs. I made an imp that had a key that they needed. He offers to sell the key to them, for a certain amount of gold. The kicker is that the gold he asks for is always 1gp more than the entire party has....

So, we need a function for finding the total party gold. Obviously, this will be an integer. There is a function to get the gold from one pc, so we'll just loop through all the pcs, check if they are in the party, and if so add the gold to the count.

Here is my version of the function, then:

NWScript:

```
// This function will return the total gold of all pcs in the
// same
// party as oPC. It will return 0 if the input is not a player
// character.
//
int GetPartyGold(object oPC)
{
if (!GetIsPC(oPC)) // if not a pc, return 0
return 0;
else // loop to get the gold
{
object oCharacter=GetFirstPC();
int nGoldCount=0;
while (oCharacter != OBJECT_INVALID)
{
if (GetFactionEqual(oPC, oCharacter)) // different parties
have different factions
nGoldCount=nGoldCount+GetGold(oCharacter);
oCharacter=GetNextPC();
}
return nGoldCount;
}
}
```

To test this, go ahead and put an npc into your module. Make a conversation for it, getting as detailed as you want. At some point, put in a node for the npc to offer to sell the key. Make the text:

“Of course I will sell you the key, mortal. It shall cost you a mere <CUSTOM1000> gold coins.”

Then, on the “ActionsTaken” tab, put in the following script:

NWScript:

```
//
// <Delete this line, cut and paste the GetPartyGold function
// here.>
//
//
void main()
{
object oTalker=GetPCSpeaker(); // Get the speaking pc
int nPrice=GetPartyGold(oTalker)+1; // Set the price to 1 more
than party gold
SetCustomToken(1000,IntToString(nPrice)); // Set the token for
the conversation
}
```

The SetCustomToken is an interesting function... it is only used inside a conversation. Basically, it allows you to stick any kind of text into a conversation. Note in the text of the conversation, we put <CUSTOM1000>. When the conversation is “spoken”, that will be taken out and replaced with whatever we have set as custom token number 1000.

Thus, that last line of the script is saying to convert the price to a string, and store that price into token 1000.

A note about custom tokens: It is best if you always set the custom token where you are going to use it. While it is possible to set it and refer back to it later, this is generally considered bad form. (*Especially* when dealing with erfs.) Further, note that tokens 0 through 9 are used by BioWare, and attempting to use them in your conversation can result in unexpected behavior, so it is best to avoid those numbers.

Default Inputs

Here is a very generic little function that is quite handy in many situations.

NWScript:

```
// This function flips a lever (or some other placeables) between
two
// positions. The STATE variable on the lever is set to 1 the
// first time
// the lever is used, and 0 when used again.
//
// Written by Celowin
// Last Updated: 7/23/02
//
void FlipSwitch(object oLever=OBJECT_SELF)
{
// STATE will be 0 for off, 1 for on
int nUsed=GetLocalInt(oLever, "STATE");
if (nUsed == 0)
```

```

    {
        AssignCommand(oLever,
        ActionPlayAnimation(ANIMATION_PLACEABLE_DEACTIVATE));
        SetLocalInt(oLever, "STATE", 1);
    }
else
    {
        AssignCommand(oLever,
        ActionPlayAnimation(ANIMATION_PLACEABLE_ACTIVATE));
        SetLocalInt(oLever, "STATE", 0);
    }
}

```

Basically, all this does is takes in as input a placeable (like a lever), and causes it to flip between two states... on and off.

Nothing in this script is all that exciting, but there is a new concept inside the declaration.

```
void FlipSwitch(object oLever=OBJECT_SELF)
```

What is the “=OBJECT_SELF” doing there? Basically, it is saying that if no input is given to the function, it is going to assume that the input is OBJECT_SELF.

So, if we have a script on a lever, we could just put the command:

```
FlipSwitch();
```

into the script, and it would to the animation and change the state of the lever which the script was attached to.

If, on the other hand, we wanted the switch to flip under some other circumstance (say when a chest was opened), we could put into the script something like:

```
oRemoteLever = GetObjectByTag("Lever6");
FlipSwitch(oRemoteLever);
```

You never *have* to declare this kind of default input, and in fact I would be kind of careful about when I would do so. However, under the right circumstances, it adds another layer of versatility to what you can do with functions.

Another Example

Here’s another example of default inputs. I like to add a lot of horror and mystery to my adventures. Most of the time, I go for subtlety, keeping the players guessing as to what is really going on. However, sometimes nothing can beat a good dose of old fashioned gore.

Hence, I use the following function once in awhile:

NWScript:

```
// This function kills a creature in an explosion of gore.
//
// oVictim is the target of the effect
// If bAffectPlot is TRUE, then the script will work on
// creatures with the plot flag set.
//
// Written by Celowin
// Last Updated: 7/23/02
//
void BloodExplode(object oVictim=OBJECT_SELF, int
bAffectPlot=FALSE)
{
if ((!GetPlotFlag(oVictim) || bAffectPlot) &&
(GetObjectType(oVictim)==OBJECT_TYPE_CREATURE))
// If the victim doesn't have the plot flag, it works
// If bAffectPlot is TRUE, it works
// Only works on creatures
{
// Create the effects: the explosion, and the death
effect
eBloodShower=EffectVisualeffect(VFX_COM_CHUNK_RED_LARGE);
effect eDeath=EffectDeath();
// Make sure the victim can be killed
SetPlotFlag(oVictim, FALSE);
// Apply the effects
ApplyEffectToObject(DURATION_TYPE_INSTANT, eDeath, oVictim);
ApplyEffectAtLocation(DURATION_TYPE_INSTANT, eBloodShower,
GetLocation(oVictim));
}
}
```

In this function, we have two inputs: the creature we want to explode (defaults to OBJECT_SELF) and a true/false input that says whether we wish it to affect creatures with the plot flag set (defaults to FALSE).

So, any one of these declarations would be exactly the same if used in a script:

```
BloodExplode();
BloodExplode(OBJECT_SELF);
BloodExplode(OBJECT_SELF, FALSE);
```

Note that we can leave off as many variables from the end as we want, but we can't leave off an input from the front if we're going to change one of the later ones. That sentence is more confusing than the concept is, so let me try to give an example.

Suppose we want to have a zombie explode when it is hit, regardless of whether the plot flag is set. So, the first input would be OBJECT_SELF (the default) but the second input

would be TRUE (non-default). There is no way to actually make use of the default for the first input, since we're changing the second. The only way to declare this would be:

```
BloodExplode(OBJECT_SELF, TRUE);
```

Something like:

```
BloodExplode(,TRUE);
```

is not valid syntax.

The other thing from this function that I should probably talk about is the conditional, the line:

```
if ( (!GetPlotFlag(oVictim) || bAffectPlot) &&  
      (GetObjectype(oVictim) == OBJECT_TYPE_CREATURE) )
```

Unless you are really familiar with this stuff, you probably got to this line, scratched your head for a bit, looked at it again, and then gave up on figuring it out. Again, it is sort of a matter of style... I loathe nested ifs. Sometimes they are needed, but if you go overboard on them, it makes your code a nightmare to trace through. Perhaps it is the fact that I have a mathematical background rather than a programming one, but the statement I just wrote is far easier for me to follow than a bunch of ifs inside each other.

So, let's break it down. When do we want the script to actually work?

- It will only affect creatures.
- It will affect creatures without the plot flag set.
- If a creature has the plot flag set, it will still affect it if bAffectPlot is TRUE.

The first one of these is immutable. It has to be a creature, otherwise nothing else matters. In other words, it has to be a creature AND other conditions need to be true. The && stands for logical "and", so the script is checking that the object type of the input is creature, and also something else.

That leaves us with the part of it that reads:

```
(!GetPlotFlag(oVictim) || bAffectPlot)
```

The || is "or". If either side of it is true, then this part of the statement is true. bAffectPlot is the input we gave... TRUE or FALSE. If it is true, we don't have anything more to worry about. The !GetPlotFlag(oVictim) is saying that if oVictim does *not* have the plot

flag, it will be true. (The ! in front sort of “flips” what we are looking for. Without the !, it would be true if oVictim *did* have the plot flag.)

So, putting it all together... it has to be a creature, AND either that creature doesn't have the plot flag OR the function is set to affect things with the plot flag. Confusing, but if you learn to use the logical operators, your code will be much cleaner.

Function Libraries

Over and over again, to use our functions we have had to put them into the script where we were going to use them. While they can still be useful, this is kind of limiting.

In particular, both the FlipSwitch and BloodExplode are functions that you might want to use over and over again, in all sorts of places in the module. (Well, maybe most people wouldn't use the BloodExplode as much as I do, but anyway...) Sure, we can cut and paste again and again, but there is a better way.

Before I begin, just a general note: I only put things into a library like this if I'm going to be using them in many different scripts. If I'm only going to use a function in one place, I won't worry about it at all. If I'm going to be using that function in two or three scripts, I'll probably just cut and paste it to where I'm going to need it. It is only the ones that I use repeatedly that I end up putting into a library.

Let's go ahead and set up a simple library to use in our module. Edit a new script, call it something like “tm_myfunctions”. Cut and paste both the FlipSwitch function and the BloodExplode function into it. Save it.

When you try to save, it will pop up with something like NO FUNCTION MAIN() IN SCRIPT. Just ignore it.

Now for the cool part. Now, in any script we write, we can put at the top the line

```
#include "tm_myfunctions"
```

And we can use any function we've put into that file!

So, as a simple example, suppose I want to pull a lever and make the nearest zombie explode. (I'll give the zombie the tag ZOMBOMB, and the lever the tag ZOMBLEV.) The script is then simplicity in itself:

NWScript:

```
// tm_zomblev_ou
// This script will destroy the nearest zombie with tag ZOMBOMB
to the lever.
//
// Written by Celowin
// Last Updated: 7/23/02
```

```

//
#include "tm_myfunctions"
void main()
{
FlipSwitch();
if (GetLocalInt(OBJECT_SELF, "STATE")==1)
{
object oZomb=GetNearestObjectByTag("ZOMBOMB");
BloodExplode(oZomb);
}
}

```

Basically, all the #include does is take the text of the file and stick it in place of itself. In essence, it does the “cut and paste” for you.

Warnings about Libraries

As amazingly useful as libraries are, there are a few things you have to be very careful about.

First, I’m going to reiterate what I said before: only put functions that you are going to use and reuse into your libraries. The larger your libraries get, the longer they take to compile. In a big module, this gets annoying *very* fast.

Even if you’re only putting in repeatedly used functions, you might find a library getting large and clunky. If so, consider breaking it down into smaller libraries: one for creatures, one for placeables, one for doors... or however else you want to divide it. It is up to you, whatever helps *you* to know what library to use for the function you want.

Along with that, also realize that since the include statement is just sticking in text, there is no reason why you have to restrict yourself one include statement in your script.

```

#include "tm_creaturelib"
#include "tm_doorlib"

```

would be perfectly acceptable.

Finally, a very important point that many non-programmers forget about... if you modify your library, you should recompile any script that includes that library. This could get ugly, since it is very easy to lose track of what scripts call your libraries. Luckily, there is a way around it.

In the toolset, there is a “Build” button. This basically says “recompile all of the scripts for this module.” Hence, if you ever modify a library, you should immediately hit that Build button to make sure everything “checks out.” Again, you will get error messages that the libraries themselves have no void main() in them... but as long as that is the only error message you get, you’re set.

Onward

This lesson has been *huge* and has a lot of little concepts. I want to stress for the people reading that if you don't understand everything in the lesson, you're probably not hurting yourself too badly. While the things here can certainly make your scripting lives easier, none of them are strictly *needed*. Definitely attempt to learn it, and ask questions if you're having problems, but if you feel you're in over your head it isn't the end of the world.

I think my next lesson is going to be on henchmen, if I can figure it out. I really don't think there is a lot of actual scripting that has to be done to set up a henchman, but I'm sick of seeing 50 questions a day on the subject. 🤔

After that, I'm going to change the format of the lessons. You really have almost all the general syntax needed for scripting at this point, so there really isn't a whole lot more I can do in that regard. Instead, I'm going to focus more on the "scripting process." I'll start with an idea, talk about design considerations, limiting the scope, and implementation. (I'll probably go light on the debugging phase, because the forum moderators wouldn't appreciate the amount of profanity that is involved in my debugging methods.)

Celwin's Scripting Tutorial Lesson X - Henchmen

Introduction

The purpose of this sequence of lessons is to take a complete beginner to programming, and teach him or her how to use NWScript to write modules. The early lessons will be very basic, and anyone that has done any coding at all will be able to skip over them. The goal here is to make the lessons so that even the people that just shudder at any type of code can learn.

Feel free to post these lessons on any forum, print them out, or modify them. However, just give me credit for doing them.

Any comments on these lessons, good or bad, can be sent to me at james.foxglove@verizon.net

I am going to assume that anyone looking at these lessons has at least played around with the Aurora Toolset a bit. If there is enough feedback that people don't know how to do the simple placements that I have in these lessons, I will consider spelling out in more detail what needs to be done.

For those of you that missed earlier lessons and are interested in looking them up, they have all been placed into the forum Scripting FAQ:
[Scripting FAQ](#)

A Few Notes on this Lesson

This lesson is a lot different from all my previous ones. In fact, I was a little hesitant to write this, because there isn't really all that much *scripting* involved with this. Most of it is just knowing where to use the functions already written by BioWare. However, there are *so* many questions on doing this that I decided to cave in.

I rewrote BioWare's function for leveling up henchmen. I did this for a few reasons:

- I wanted something more generic. There are limits inherent in BioWare's function that are difficult to change.
- I didn't like the naming convention needed for BioWare's function. It doesn't make a lot of sense to me.
- Boredom is a powerful motivator. Most of the process of creating a henchman is pure tedium, and I wanted to do something interesting.
- Er, well... I couldn't find BioWare's function at first. I pretty much had the whole thing written at the point where I found the right place to look.

Also, please note that putting a henchman into a module is *not* a trivial task, even with

step by step instructions. Expect to spend a few hours getting even a basic one set up, and a lot more if you want to give them interesting dialogue.

Other people have worked on extensions to henchmen... improving the AI, flexibility in equipping them, and multiple henchmen are some of the ones that I've heard about. I make *no claims whatsoever* that this lesson will mesh with anyone else's scripts. I'm making a basic henchman here, any modifications you wish to make are not my responsibility.

Step 1: Create the Lowest Level Version

The first thing to do is to create the "first" henchman, at the lowest level. (For the official campaign, the lowest level henchmen are level 4, but there is no reason here why you need to stick to that.)

My suggestion on handling this (and all higher level versions) is to start out with a pc. Create it using the "Create New Character" in the game, and then put the character into a module with a "level up" lever.

Create a simple module, with one tiny area. Insert a placeable lever, and into the "On Used" script, attach the following:

NWScript:

```
// On Used Script: tm_xplever_ou
// This function awards the lever puller with tons of xp.
// To be used only for testing purposes, remove from final
// module.
//
// Written By: Celowin
// Last Updated: 7/25/02
//
void main()
{
object oPC=GetLastUsedBy();
GiveXPToCreature(oPC, 10000);
}
```

Save everything, and start up the module with your "pc version" of the henchman. Pull the lever, and start leveling up your character. Pick all the skills, spells, feats, etc. as you feel fit the character. Then, when you have the "henchman" to the appropriate level for the first version, stop and write everything down.

Remember to save the game (we'll be coming back here later), and then open up your "real" module, where you want your henchman, in the toolset. Create your henchman, giving it all the abilities that we figured out using the xp lever.

From here, on, to stop using the term "your henchman," I'm going to be using the name

“Coric Galroud.” Of course, there is nothing magical about this name, you can call it whatever you want. I do, however, suggest you keep the first name relatively short, about 10 characters maximum.

Important! Badger Alert! I can’t count the number of times I’ve seen a message like this, “Help! I made a henchman, and when it leveled up it turned into a badger!” In almost every single case, this is because the tag and/or blueprint for the npc was set wrong.

Again, I’m not using the BioWare level up functions, so my naming conventions are a little bit different from theirs. The tag for the henchman should be the first name, but in all caps. In my case, the tag is CORIC. The blueprint (found on the Advanced tab) is the name in all lower case, plus the two digit version of the henchman level. Again, in my case for a level 4 Coric, the blueprint would be coric04.

I can’t stress this enough. Tag = uppercase, blueprint = lowercase with level.

One more time –

Tag: CORIC

Blueprint: coric04

Moving on, other things to set on our henchman:

- Make sure the faction is set to “Merchant”
- Make sure the No Permanent Death box is checked.
- Equip the henchman. Start him out with a melee weapon equipped, but have a ranged weapon in inventory, with plenty of ammunition.
- There will be lots of conversation with the henchman. Make certain it has a portrait.
- Make sure perception range is set to Default.

Now, we need to set up the basic scripts for our henchman. I’m just going to use a “plug and play” approach here, we’re not going to delve into exactly how these scripts work. All these scripts can be selected from the drop down menu, or else the names typed in manually. (Of course, double check you have the name exactly right if you type it.)

A couple of notes: While BioWare references nw_ch_aca for the OnRested script, this script doesn’t seem to exist. I left it blank, and I don’t really notice any difference in behavior. Further, the nw_ch_acd script does absolutely nothing. I put it in for reference, but you can safely leave it out if you’re not planning on modifying your henchman behavior.

- OnBlocked: nw_ch_ace
- OnCombatRoundEnd: nw_ch_ac3
- OnConversation: nw_ch_ac4
- OnDamaged: nw_ch_ac6
- OnDeath: nw_ch_ac7
- OnDisturbed: nw_ch_ac8
- OnHeartbeat: nw_ch_ac1
- OnPerception: nw_ch_ac2
- OnPhysicalAttacked: nw_ch_ac5
- OnRested: Leave Blank
- OnSpawn: nw_ch_ac9
- OnSpellCastAt: nw_ch_acb
- OnUserDefined: nw_ch_acd

Ok, check one more time that your tag and blueprint are set properly, and also make certain that the henchman actually has the feats to use the equipment you've given it. Once everything is set, it is time to work on the conversation.

Step 2: Setting up the Conversation

Actually, before going too much further, I would ok out of the npc, and save the module. There is no sense in losing all of your hard work if the editor crashes. After that, start back up, open the npc, and edit the conversation.

This is probably the most complicated part, because there are *lots* of annoying little details for the conversation. I stripped the conversation down about as far as it could go, down to just "do this," and there are *still* a lot of nodes.

Anyway, be sure you have a bit of time to spend setting this up, and edit the tree to look like this (best I can do with ascii). Note that the H3: type things are just so I can reference specific conversation nodes, and should not be typed in. H is something the henchman is saying, P is something the player is saying.

NWScript:

```

Root
|-H1: Greetings, do you have need of a henchman?
|
|   |-P1: Yes, join me.
|   |   |-H2: Excellent, that I shall.
|   |   |-P2: Not at this time.
|   |       -H3: As you wish. Speak to me if you change your mind.
|
|-H4: I'm sorry, I serve another and have no time to speak with
you.
|
|-H5: How may I serve you, <master/mistress>?
|   |-P3: I no longer have need of your services.
```

| -H6: I'm sorry you feel that way, but I shall leave if that
is your desire.

| | -P4: Aye, it is.

| | -P5: Aw, if it makes you feel that badly, you can stay.

| -P6: I wish you to undergo some training.

| | -H7: Absolutely! I am always eager to learn.

| -P7: Let's discuss our adventuring plans.

| | -H8: How should I behave?

| | | -P8: I want you to fight hand to hand.

| | | - H9: Done. Anything else?

| | | | - P9: Yes.

| | | | - (Link to H8)

| | | | - P10: No.

| | -P11: I want you to fight from range.

| | | - (Link to H9)

| | -P12: Change your following distance.

| | | -H10: Aye, aye. How close do you want me?

| | | | -P13: Stick close to me.

| | | | - (Link to H9)

| | | | -P14: Keep a moderate distance.

| | | | - (Link to H9)

| | | | -P15: I don't want to be seen with you. Stay far
away.

| | | | - (Link to H9)

| | -P16: I think you need to change your healing
strategies.

| | | -H11: You're the boss. When should I heal?

| | | | -P17: Heal when I'm slightly wounded. (75%)

| | | | | - (Link to H9)

| | | | -P18: Heal when I'm moderately wounded. (50%)

| | | | | - (Link to H9)

| | | | -P19: Wait until I'm near death. (25%)

| | | | | - (Link to H9)

| | -P20: Stop searching all the time! It is getting on my
nerves.

| | | - (Link to H9)

| | -P21: We're missing important clues. I think you should
search.

| | | - (Link to H9)

| | -P22: Let's try to surprise the enemy. You sneak
around.

| | | - (Link to H9)

| | -P23: I favor the direct approach. Stop sneaking.

| | | - (Link to H9)

| | -P24: I'm inept with mechanical things. Help me with
doors and chests.

| | | - (Link to H9)

| | -P25: Opening doors and chests is a tricky business.
Let me handle them.

| | | - (Link to H9)

| | -P26: That's all for now.

Note, of course, that I would never actually use a conversation this stripped down in an actual adventure. Your henchman would have zero personality. Also, I haven't put any provisions for "memory" of the henchman.

Ok, the basic tree is set up, now we have to attach conditions and actions to it. Again, I'm just going to use the BioWare functions as much as possible. A few places I've made my own, but I'll deal with those last. (Note: A few of the BioWare functions reference journal updates. While it would be good form to comment those lines out, it isn't strictly necessary. Just be aware that if you "Build" your module, the editor will throw a hissy fit over the missing journal entries. Ignore it, the module will still run fine.)

- H1, Appears When: tm_hirecheck (see below)
- H2, Actions Taken: nw_ch_action_13
- H4, Appears When: tm_othercheck (see below)
- P4, Actions Taken: nw_ch_remove
- P6, Appears When: tm_levcheck (see below)
- H7, Actions Taken: tm_henchlev (see below)
- P8, Appears When: nw_ch_comrange
- P8, Actions Taken: nw_ch_gomelee
- P11, Appears When: nw_ch_commelee
- P11, Actions Taken: nw_ch_goranged
- P13, Actions Taken: nw_ch_dist_6
- P14, Actions Taken: nw_ch_dist_12
- P15, Actions Taken: nw_ch_dist_18
- P16, Appears When: nw_ch_heal_0
- P17, Actions Taken: nw_ch_heal_75
- P18, Actions Taken: nw_ch_heal_50
- P19, Actions Taken: nw_ch_heal_25
- P20, Appears When: nw_ch_yes_srch
- P20, Actions Taken: nw_ch_srch_off
- P21, Appears When: nw_ch_no_srch
- P21, Actions Taken: nw_ch_srch_on
- P22, Appears When: nw_ch_no_stlth
- P22, Actions Taken: nw_ch_stlth_on
- P23, Appears When: nw_ch_yes_stlth
- P23, Actions Taken: nw_ch_stlth_off
- P24, Appears When: nw_ch_no_locks
- P24, Actions Taken: nw_ch_lock_on
- P25, Appears When: nw_ch_yes_locks
- P25, Actions Taken: nw_ch_lock_off

Step 3: Writing Our Functions

Ok, there were a few functions that we need up there for the conversation. All but the level up one is very easy, so I'll start with those.

For node H1: tm_hirecheck

NWScript:

```
// Starting Conditional script: tm_hirecheck
// Should only be placed in a henchman conversation file.
// This function returns TRUE if the henchman does not have a
master.
//
// Written by: Celowin
// Last Updated: 7/25/02
//
int StartingConditional()
{
    int iResult;
    iResult = GetMaster()==OBJECT_INVALID;
    return iResult;
}
```

For node H4: tm_othercheck

NWScript:

```
// Starting Conditional script: tm_othercheck
// Should only be placed in a henchman conversation file.
// This function returns TRUE if the pc speaking is not the
henchman's
// current master.
//
// Written by: Celowin
// Last Updated: 7/25/02
//
int StartingConditional()
{
    int iResult;
    iResult = GetPCSpeaker()!=GetMaster();

    return iResult;
}
```

Now for the hard stuff. I'm going to be using a few functions multiple times, so I'm going to write a library. People that went through lesson 9 should know what these are. If you are just coming here to learn about henchmen, a quick rundown: Put in the following as a script tm_henchlib, and save it, not attached to any event handle. It won't compile properly, you'll get an error message like NO VOID MAIN() IN SCRIPT. Just ignore that message.

A further note. I've made it so that it is easy to change three properties of henchmen: the minimum level, the maximum level, and how many levels behind the pc the henchman will "lag."

For example, say you're writing a module for characters of level 4-10. You want to give

your characters a bit of a boost early on, so you set the minimum henchman level to 6. Later on, you want less of an impact from henchmen, so you set that they will lag 2 levels behind the pcs. And just in case the pcs get more xp than you anticipate, you set the maximum level on the henchman to 10.

In that case, when a level 4 pc gets the first henchman, it will start out at level 6. It will remain level 6 until the pc hits level 9... then the henchman becomes level 7. The henchman will always remain 2 levels behind the pc until the pc hits level 12. From then on, the henchman will always be level 10.

Anyway, to set these things, just alter the numbers at the start of the file.

NWScript:

```
// My henchman routines: tm_henchlib
//
// Written by: Celowin
// Last updated: 7/25/02
//
#include "nw_i0_henchman"
//
// Defines how many levels behind the pc the henchman lags
// Minimum of 0
int HENCH_LAG=1;
// Defines the lowest level the henchman can be
// Minimum of 1, Maximum of 20
int HENCH_MIN=4;
// Defines the highest level the henchman can be
// Minimum of HENCHMIN, Maximum of 20
int HENCH_MAX=14;
//
// This function figures out what level the henchman
// "should" be, based on the min, max, and lag
// numbers above, as well as the pc level.
int GetTargetLevel(object oHench=OBJECT_SELF)
{
// Find the pc's level.
int nMasterLevel=GetHitDice(GetMaster(oHench));
// Apply the lag.
int nTargetLevel=nMasterLevel-HENCH_LAG;
// If that is less than the minimum level, the
// henchman should be that minimum level.
if (nTargetLevel < HENCH_MIN)
    nTargetLevel = HENCH_MIN;
// If the level is over the max, cap the level
// to that max.
if (nTargetLevel > HENCH_MAX)
    nTargetLevel = HENCH_MAX;
return nTargetLevel;
}
//
// This function checks to see if the henchman
// is ready to level up.
int GetReadyToLevel(object oHench=OBJECT_SELF)
```

```

{
// Find both the current henchman level, and
// what level the henchman should be.
int nTargetLevel=GetTargetLevel(oHench);
int nCurrentLevel=GetHitDice(oHench);
// If the current level is too low, it is ready
// to level up.
if (nCurrentLevel<nTargetLevel)
    return TRUE;
else
    return FALSE;
}
//
// This is the biggie. This is the function to
// actually level up the henchman.
void LevelHench(object oHench=OBJECT_SELF)
{
// Find the desired level.
int nTargetLevel=GetTargetLevel(oHench);
// Whee! Fun with string manipulation!
// All this is working to find the appropriate blueprint
// to create the "new and improved" henchman.
string sNewBlueprint=GetTag(oHench);
sNewBlueprint=GetStringLowerCase(sNewBlueprint);
if (nTargetLevel<=9)
    sNewBlueprint=sNewBlueprint+"0";
sNewBlueprint=sNewBlueprint+IntToString(nTargetLevel);
// Create the new henchman.
// There will be a "fade in" of the new one and a "fade out"
// of the old.
object oNewHench=CreateObject(OBJECT_TYPE_CREATURE,
sNewBlueprint, GetLocation(oHench));
// Get rid of the old henchman. You're fired!
object oMaster=GetMaster(oHench);
RemoveHenchman(oMaster, oHench);
// Copy our desired behavior patterns to the new henchman.
CopyLocals(oHench, oNewHench);
AssignCommand(oNewHench,SetAssociateListenPatterns());
// Add in our new henchman.
AddHenchman(oMaster,oNewHench);
// Take out the trash.
DestroyObject(oHench);
}

```

Now that we have that library set up, the rest of the scripts are easy, just calling the appropriate functions from the library.

For node P6: tm_levcheck

NWScript:

```

// Starting Conditional script: tm_henchlev
// Should only be put into a conversation file for a henchman
//
// Returns TRUE if the henchman is ready to level up.

```



```

//
#include "tm_henchlib"
//
int StartingConditional()
{
    int iResult;
    object oHench=GetHenchman(GetPCSpeaker());
    iResult = GetReadyToLevel(oHench)==TRUE;
    return iResult;
}

```

For node H7: tm_henchlev

NWScript:

```

// Action Taken script: tm_henchlev
// To be put in a henchman conversation.
//
// This amazingly complicated script causes a henchman to level
up.
//
#include "tm_henchlib"
//
void main()
{
    LevelHench();
}

```

Ok, enough for the conversation. Save the conversation, and save the module.

Step 4: Creating Higher Level Versions of the Henchman

At this point, our “base” henchman is completely done. It has equipment, stats, *the appropriate tag and blueprint*, and a conversation. Now, we need to create all the henchman that it will “level up” into.

Go back to the save game from when we first found the stats for our henchman. Now, using the lever for xp, keep levelling up, and write down the changes as you go. You don’t need to write down absolutely everything, but at least write down the information from the “changes” window at the end of the level up process.

For example, if I’m working on a fighter henchman, I might write down something like:

Lev 4: Str +1, HP +14, Disc +1, Parry +1, Spec(longsword)

And so on for each level up, until I’m at the max henchman level.

Now, close out the game, and start up the toolset.

Go to your henchman, and add him to your custom palette.

On the palette, select “Edit Copy.” Start putting in changes to the next higher level. Update the character level, feats, skills... everything you wrote down from before.

Warning! Badger alert! Remember to update the blueprint to the appropriate thing. You shouldn’t have to change the tag, but the blueprint needs to be changed to the “coric05” format. Also, remember that you have to change the class level for your henchman to account for the new level.

Yet another reminder –

Tag: CORIC

Blueprint: coric12

Keep doing this. Each time you have the new level in, edit another copy and update it to the next level, until you have all your versions set up.

Now, go back through the henchmen and play with their equipment, giving them more level appropriate stuff. Starting at level 7, their ranged weapons should have the “Unlimited Ammunition” ability. (Unfortunately, that makes the weapon level 7 minimum, so it can’t be given to lower level henchmen.)

Stats, spells, skills, feats, levels, *blueprint*, and equipment. Remember to change all these things for each new copy of the henchman. Then, we’re almost done.

Step 5: Module Properties

Almost everything is set to go at this point. Just a few minor things to add to the module to make things work.

First, at whatever “home base” place you want in your module, add a waypoint with tag NW_DEATH_TEMPLE. This is where the henchman will return to when it dies.

Second, we need to add a script to have the henchman level up when the pc does. Go to Edit -> Module Properties -> Events -> OnPlayerLevelUp, and put in the following script:

NWScript:

```
// OnPlayerLevelUp script: tm_levelup
//
// When a player levels, this checks to see if the henchman is
// ready to
// level up as well, and takes necessary action.
//
#include "tm_henchlib"
//
void main()
```

```
{
// Find the leveling pc and its henchman
object oPC=GetPCLevellingUp();
object oHench=GetHenchman(oPC);
//
if (GetIsObjectValid(oHench)) // Don't go on if no henchman
{
    if (GetReadyToLevel(oHench)) // Don't go on if not time to
level
    {
        LevelHench(oHench);
    }
}
}
```

Conclusion

And that should do it. If you've followed everything, your henchman should pretty much behave exactly like the ones in the official campaign. (No story or quests, of course. You'll have to come up with those on your own.) You have at least a bit more flexibility on henchman level up behavior if you want to exercise it.

Starting next lesson, I'm going to be taking a slightly different tack. I've discussed my plans in a few places, I won't reiterate them here. Anyway, we'll see how it goes, and whether people will like the new format.

Now that all the real basics are covered, I don't feel quite the pressure to churn out lessons. I'll continue to write, but probably at a lower pace. I'm thinking maybe one a week... since Wednesdays we've been getting new stuff from BioWare, maybe I'll try to schedule putting out a new lesson every Friday. That way you have something to play with over the weekend.

Celowin's Scripting Tutorial Lesson XI - Process Example: Bar Brawl

Introduction

The purpose of this sequence of lessons is to take a complete beginner to programming, and teach him or her how to use NWScript to write modules. The early lessons will be very basic, and anyone that has done any coding at all will be able to skip over them. The goal here is to make the lessons so that even the people that just shudder at any type of code can learn.

Feel free to post these lessons on any forum, print them out, or modify them. However, just give me credit for doing them.

Any comments on these lessons, good or bad, can be sent to me at james.foxglove@verizon.net

I am going to assume that anyone looking at these lessons has at least played around with the Aurora Toolset a bit. If there is enough feedback that people don't know how to do the simple placements that I have in these lessons, I will consider spelling out in more detail what needs to be done.

For those of you that missed earlier lessons and are interested in looking them up, they have all been placed into the forum Scripting FAQ:
[Scripting FAQ](#)

A Word of Thanks

I would like to send out my heartiest thanks to Rhodan for helping me with this lesson. His aid was invaluable in discussing the scripts, and helping to figure out all the difficulties in manipulating reputations.

Suggestion for Review

This lesson makes extensive use of loops (Lesson 6) and user defined events (Lesson 4). If you are at all shaky on those concepts, I suggest you go back and take a look at them again. If not, the scripts in this lesson are likely to be way over your head.

The New Format

This is the first of the "new" lessons. Lessons 1 through 9 taught all the basics of scripting. Lesson 10 was sort of a "filler" episode. At this point, we basically have all the tools, it is just a matter of figuring out how to use them.

Something that has come up over and over in the feedback on my lessons is that people may understand the commands, but they don't know *when* to use them. There is actually an art to designing a properly functioning script. Like any art form, some people have

more natural talent than others... to some it comes naturally, while others have to struggle with it.

However, and I can't stress this enough: *anyone can learn*. Just because you are trying to do some complicated script, and don't see in 5 seconds how to do it, doesn't mean that you will never be able to.

In an effort to teach this process of scripting, the way I present the lessons is going to change. I'm not going to focus on the individual commands any more, unless there is some strange one that I pull out of nowhere. Instead, I'm mainly going to try to explain the *process* that I go through in order to get a script working. I'm going to try to document everything: the original idea, the design process, the problems I encounter. In the end, I'm hoping that we come up with a neat, useful script.

Some people will just rip the scripts out of the lessons and ignore the rest, I'm sure. That is fine... but I'm hoping that some people will actually read through it, and take some understanding of how to go about creating their own scripts.

I'm going to start with a complicated setup... by far the most ambitious script that I have created to date. I'm doing this partially to challenge myself, but I'm also doing it so that you can see all the careful thought and planning that goes into an involved project.

At any rate, ladies and gentlemen, let me introduce you to the...

Singing Spider Tavern

I've never been one to go to any of those high-class bars. I hear they serve their drinks in glass mugs, and I have to laugh. A glass mug wouldn't last one night at the Singing Spider Tavern, and I can't imagine anyone who would drink from one would either. The Spider draws a rough crowd... dock workers, street toughs, miners... the kind of people who ain't got nothing better to do at night than to get drunk and take out their frustrations at life on one another. My kind of people.

The next several lessons are going to be spent trying to set up different aspects of the Singing Spider Tavern. At the end, we'll have a lively place with a lot of character, perfect for launching any number of adventures. For right now, we're going to focus on what happens when things get out of hand. Basically, we want a good old fashioned, all in good fun bar brawl.

Original Design

Here's basically what we want to happen:

- There are three types of customers at the bar... dock workers, “Shivs” (members of a local street gang), and miners. Each group will not attack their own, but are more than willing to pound on the other groups or on the pcs.
- There are multiple ways to start the brawl. Walking up and punching someone is bound to do it, but it shouldn’t take nearly that much. The customers are always spoiling for a fight, and saying the wrong thing can easily toss the match onto the oil.
- As long as no one uses a weapon, the fight is all in good fun. No one really gets hurt, though there will be a number that end up taking a nap.
- If a pc does draw a weapon, things become much more serious. The customers all draw their own weapons, and people actually die.

Potential Problems

Looking through the list and thinking about it, I immediately see several things that we’re going to have to work to resolve. You should *always* think ahead and try to foresee difficulties. There is no way you’ll catch all of them... there are always things you didn’t imagine going wrong. However, the more you plan for from the start, the better off you will be.

- The fight shouldn’t cause “real death” unless it turns deadly. We’ll need some way of handling that.
- We need a way to determine whether weapons are being used or not.
- Spells add another layer of confusion.
- We can apply a sleep effect to the people that get knocked out... but elves are immune to sleep.
- We’re going to need to be certain to “turn off” hostilities once the fight has been resolved.
- There are going to be a *lot* of scripts running. We’ll have to be certain that they are written in such a way to minimize the load on the processor.

Basic Implementation

My first thought on getting this whole thing to work is to start with the “non-deadly” combat. How am I going to make it so that things do no “real” damage, but still can knock someone out?

I can’t just set the plot flags on people, because then there is no way of determining how much damage would have been done.

My first thought was to heal the damage as soon as it was done, and hold a local variable

on the character (pc or npc) to keep track of how much damage was done. The more I considered this, though, the more problems I saw. There is no OnDamaged event for pcs, so I'd have to do something with the heartbeat. And what if some poor weak mage wandered into the fray, and died in one hit, before any healing?

Instead, I'll just have the fight play out as normal, using real damage... but I'll modify the OnDeath and OnDying events to handle what I wish. If the fight turns deadly, I'll partially heal everyone, to account that the "pummeling damage" isn't as harmful as "real damage."

Now, those events are going to have to behave differently depending on whether the fight was deadly or not. For that matter, we'll have to be careful that for the pcs, it won't affect them if they aren't in the bar. Because we have things changing because of the "state" of the fight, immediately the first thing you should think about is a local variable to keep track of that state. I'll store it on the area.

We'll call the variable BRAWL_LEVEL. It will be 0 for peaceful (the original state), 1 for the "good clean fun" brawl, and 2 for "ugly."

One of the keys to scripting is to break down a large task into manageable steps. For right now, let's try to get this much working: No staff, we'll just have a few patrons in the bar. Originally all are peaceful, but if one is attacked the brawl starts. When one "dies" it will be knocked unconscious for a bit... let's say 2 minutes real time, corresponding to 1 hour until the default time scale. We won't worry about deadly combat yet. In fact, we won't even worry about the different types of patrons, we'll put them all into the same faction.

Now that I have a goal, I start playing with it. I try what I outlined before... when the npc dies, he is resurrected, healed a bit, and put to sleep. There are *tons* of problems that I didn't anticipate... this is actually perfectly normal. Let's see, here are some of the things that happened as I went through and tried to make it work:

- It was completely random whether the npc would "stay sleeping" or not. Sometimes they would sleep, but sometimes an accidental "extra" hit on them would wake them up.
- The npc would resurrect properly, but would "fade out" after a few seconds.
- The npc would die, stand up, and fall over again. While I could almost imagine them trying to rise up and collapsing, it just didn't look right.
- The npc's were too eager to jump back into the fray. What was happening was that while asleep, they were hearing the "yells for help" from the other patrons I attacked, and that made them hostile again. So as soon as they woke up, they would attack again.

There were other problems as well... there isn't a point in my listing any more. The point

is, that you can't expect your script to work 100% properly the first time. There are *always* things you don't expect, or things that you've forgotten about. You can't give up at the first sign of a problem, you have to jump in and see what is causing it.

For example the second point up there... I figured that the npc was being destroyed because of the death. (Just as the "corpse" of an npc is destroyed after a moment.) I knew, though, that henchmen *didn't* destroy themselves when they died. So, I plopped a henchman into my module, and looked at the OnDeath script. There was a lot of junk in there, most of it didn't look very important. Then I found that somewhere in there, the SetIsDestroyable command was used. I looked up that command, and found that one of the things set by that was whether the corpse would remain on death. I tested it, and found that it fixed my fadeout problem.

For all the other problems, I was starting to get a little bit frustrated. I tried a few things, and was getting some limited success, but it still wasn't working really the way I wanted. Then, I had an inspiration... what if, instead of resurrecting them and sleeping them, I just left them dead? I could delay the resurrection for 2 minutes. I tested it, and it actually fixed just about every problem I was having. Since the npc was dead, an extra hit on it didn't matter. It fell down, and laid there until it was ressed... so no stupid standing up/down stuff. And finally, since the npc was dead when anyone else was attacked, it wasn't hearing the shouts for help.

In fact, when I thought about it, I realized if I were to use the same method for the pcs, it would get around the "no sleeping elves" problem.

Anyway, that points out another aspect of the scripting process... don't be too attached to doing things one way. Sometimes, another approach works much better. It may not fit your original "vision" of what is happening, but sometimes a bit of compromise can result in a much better project overall.

Anyway, if you want to test this "start" of our process, here is the script I eventually ended up with. (Put this in the "On User Defined" handle, and in the OnSpawn script uncomment the lines for the OnDeath and OnDamaged user defined events.) I used basic level 3 fighters for the npcs.

NWScript:

```
// User Defined Script: fm_sspatron_ud
//
// Sets up the special behaviors for a patron at the
// Singing Spider Tavern. Specifically, death is
// a fleeting thing. The npc will get up after a
// couple of minutes. This is supposed to simulate
// getting knocked out in a brawl.
//
// Written by: Celowin
// Last Updated: 7/27/02
//
void main()
```



```

{
int nCalledBy = GetUserDefinedEventNumber();
switch(nCalledBy)
{
case 1006: // On Damaged
// At this point, this doesn't do much. Right now, we're
// just setting up for future things.
SetLocalInt(GetArea(OBJECT_SELF), "BRAWL_LEVEL", 1);
break;
case 1007: // On Death
if (GetLocalInt(GetArea(OBJECT_SELF), "BRAWL_LEVEL") == 1)
// Check if the brawl is "safe."
{
// First, make sure the npc doesn't self destruct, and
// can't be bothered during his or her "nap."
SetIsDestroyable(FALSE, TRUE, FALSE);
// A bit annoying, but this makes it so the npc won't be
// hostile when waking up.
SurrenderToEnemies();
// After 2 minutes, res the npc.
effect eRes = EffectResurrection();

DelayCommand(120.0, ApplyEffectToObject(DURATION_TYPE_PERMANENT,
eRes, OBJECT_SELF));
// Immediately after, heal the npc to half health.
int nHealAmount = FloatToInt(
IntToFloat(GetMaxHitPoints())/2);
effect eHeal = EffectHeal(nHealAmount);

DelayCommand(120.1, ApplyEffectToObject(DURATION_TYPE_PERMANENT,
eHeal, OBJECT_SELF));
}
break;
}
}
}

```

Expanding The Script

At this point, the basic idea is working (for the npcs, if not the pcs), and now we start adding more details to it. Again, the idea is to take small pieces, and get one tiny thing working at a time.

At this point, the thing that really irks me is that even if the patron were to leave the bar, he would still only be knocked unconscious. The problem is that it always sets the current area state to a bar brawl when it gets damaged, no matter where it is.

The eventual way that I decided to fix this was through the BRAWL_LEVEL variable. Instead of starting it at 0, I'll make 1 peaceful. 2 will be the brawl, and 3 will be deadly. That way, most areas will be at 0 (default), and we can easily put an extra little check to be sure we don't set it away from that.

Now, if you can avoid it, you don't want to make a shift like this *too* far into the

process... it can be difficult to track down all the changes you have to make. However, in this case, we're really just beginning, and I think it will save us a lot of headaches later on, so I go ahead and change it.

Now, the next thing is to have the brawl turn deadly if a weapon is used. We'll just add a bit to the OnDamaged part of our script, to check to see if the attacker has a weapon wielded. If so, we'll set the brawl state up to 3... and since we already check the brawl level in our OnDeath part, this means the death will be permanent.

Now for a tricky part. When the brawl goes deadly, we want all the patrons to "heal" a bit, and draw weapons. I could add this into our UserDefined script, but since this is an "Area Wide" thing, I'm going to write a separate script, for the area. I'll give it a user defined number of 3050. I'll call this function only the first time the fight goes deadly.

There were a number of things that I had difficulty with here... most of them dealing with my forgetting to give the npcs the feat needed to equip the weapon. Eventually, I got it working.

At this point, there are 4 scripts in the module:

The OnSpawn for the patrons is just the default, with the OnCombatRoundEnd, OnDamaged, and OnDeath events commented out. (I also uncommented the ambient animations, to give them a bit of life.)

The OnEnter for the area:

NWScript:

```
// On Enter Script:  fm_usspider_en
// This is the script for the area The Singing
// Spider Tavern.
//
// It does two things:
// 1. Initializes the brawl state to peaceful, if it
//    hasn't been set.
// 2. Stores the pc's current hit points on enter.
//
void main()
{
// Get the current brawl status.
int nBrawl = GetLocalInt(OBJECT_SELF, "BRAWL_LEVEL");
// If the status hasn't been set, set it to 1 (peaceful).
if (nBrawl == 0)
    SetLocalInt(OBJECT_SELF, "BRAWL_LEVEL", 1);
// Get the entering object, check if it is a pc, and if so,
// store the current hp.
object oPC = GetEnteringObject();
if (GetIsPC(oPC))
    SetLocalInt(OBJECT_SELF, GetName(oPC),
GetCurrentHitPoints(oPC));
}
```

The User Defined for the area:

NWScript:

```
// User Defined Event: fm_sspider_ud
// This script defines all the major events for the
// Singing Spider Tavern.
// Most of the events deal with the bar brawl that
// happens there.
void main()
{
int nCalledBy = GetUserDefinedEventNumber();
// Declarations up top, out of the switch.
object oCreature;
string sTag;
int nMax;
int nCurrent;
int nHealAmount;
effect eHeal;
switch (nCalledBy)
{
case 3050: // The brawl just turned deadly.
oCreature = GetFirstObjectInArea();
while (oCreature != OBJECT_INVALID)
{
if (GetObjectType(oCreature) == OBJECT_TYPE_CREATURE)
{
if (!GetIsPC(oCreature))
{
sTag = GetTag(oCreature);
if (sTag == "SSPATRON")
{
// Tell the Creature to equip a weapon at the next
chance.

SetLocalInt(oCreature, "EQUIP", 1);
// Calculate the healing amount
nMax=GetMaxHitPoints(oCreature);
nCurrent=GetCurrentHitPoints(oCreature);
nHealAmount = FloatToInt( IntToFloat(nMax-nCurrent) /
2 );

// Heal the "fake" damage
eHeal = EffectHeal(nHealAmount);
ApplyEffectToObject(DURATION_TYPE_INSTANT, eHeal,
oCreature);
}
}
else
{
// For pcs, healing but nothing else
// Heals half the difference between current and what
the
// pc had when entering the tavern
nMax = GetLocalInt(OBJECT_SELF, GetName(oCreature));
nCurrent = GetCurrentHitPoints(oCreature);
nHealAmount = FloatToInt( IntToFloat(nMax-nCurrent) / 2
```

```

);
    // Apply the heal
    eHeal = EffectHeal(nHealAmount);
    ApplyEffectToObject(DURATION_TYPE_INSTANT, eHeal,
oCreature);
    }
    }
    oCreature = GetNextObjectInArea();
    }
    break;
}
}
}

```

And finally, the user defined for the patron:

NWScript:

```

// User Defined Script: fm_sspatron_ud
//
// Sets up the special behaviors for a patron at the
// Singing Spider Tavern. Specifically, death is
// a fleeting thing. The npc will get up after a
// couple of minutes. This is supposed to simulate
// getting knocked out in a brawl.
// If the fight has turned deadly, death behaves
// pretty much as normal.
//
// Written by: Celowin
// Last Updated: 7/28/02
//
void main()
{
int nCalledBy = GetUserDefinedEventNumber();
// Even though only used in part of the switch,
// These need to be declared up top.
int nBrawl;
object oAttacker;
object oRWeapon;
object oLWeapon;
switch(nCalledBy)
{
case 1003: // On Combat Round End
    if (GetLocalInt(OBJECT_SELF, "EQUIP")==1)
    {
        // If the fight just turned deadly, need to equip a weapon
        // Create a dagger first
        oRWeapon = CreateItemOnObject("nw_wswdg001",OBJECT_SELF);
        // Equip the dagger
        ClearAllActions();
        ActionEquipItem(oRWeapon, INVENTORY_SLOT_RIGHTHAND);
        // Remind the npc that he has equipped already
        SetLocalInt(OBJECT_SELF, "EQUIP",0);
    }
    break;
case 1006: // On Damaged
    // If the brawl level is 1 (peaceful), immediately set it

```

```

// up to 2 (safe brawl).
nBrawl = GetLocalInt(GetArea(OBJECT_SELF), "BRAWL_LEVEL");
if (nBrawl == 1)
{
    nBrawl = 2;
    SetLocalInt(GetArea(OBJECT_SELF), "BRAWL_LEVEL", 2);
}
// Next, if the attacker wielded a weapon, set the brawl
// state up to 3 (deadly). Note that torches and shields
// are considered out of place in a brawl, and will set it
// off.
if (nBrawl == 2)
{
    oAttacker = GetLastAttacker();
    oRWeapon = GetItemInSlot(INVENTORY_SLOT_RIGHTHAND,
oAttacker);
    oLWeapon = GetItemInSlot(INVENTORY_SLOT_LEFTHAND,
oAttacker);
    if (GetIsObjectValid(oRWeapon) ||
GetIsObjectValid(oLWeapon))
    {
        SetLocalInt(GetArea(OBJECT_SELF), "BRAWL_LEVEL", 3);
        SignalEvent(GetArea(OBJECT_SELF), EventUserDefined(3050));
    }
}
break;
case 1007: // On Death
if (GetLocalInt(GetArea(OBJECT_SELF), "BRAWL_LEVEL")==2)
// Check if the brawl is "safe."
{
    // First, make sure the npc doesn't self destruct, and
    // can't be bothered during his or her "nap."
    SetIsDestroyable(FALSE, TRUE, FALSE);
    // A bit annoying, but this makes it so the npc won't be
    // hostile when waking up.
    SurrenderToEnemies();
    // After 2 minutes, res the npc.
    effect eRes = EffectResurrection();

    DelayCommand(120.0, ApplyEffectToObject(DURATION_TYPE_PERMANENT,
eRes, OBJECT_SELF));
    // Immediately after, heal the npc to half health.
    int nHealAmount = FloatToInt(
IntToFloat(GetMaxHitPoints())/2);
    effect eHeal = EffectHeal(nHealAmount);

    DelayCommand(120.1, ApplyEffectToObject(DURATION_TYPE_PERMANENT,
eHeal, OBJECT_SELF));
}
else
    SetIsDestroyable(TRUE); // Otherwise, "real" death
break;
}
}

```

Ok, at this point I can see that I can't go on posting the scripts every time I change something. I'll just hold off on putting any more up until the end.

Adding Different Groups

I really thought that adding the different groups to the brawl would be very simple.... oh, how wrong I was! I ended up fighting this for over two weeks, off and on.

It was here that I called in Rhodan. Mainly, I wanted someone to bounce ideas back and forth with... often, by discussing the problem, you see something you hadn't noticed before. The first several problems, I ended up figuring out solutions just by trying to describe what was going on. Of course, the later ones were much more difficult, and I lost a lot of hair over them.

The basic idea is simple, and really didn't change in implementation. A lot of details were modified, but the overall attack on the problem was never altered.

We have three factions: Miner, Dockworker, and Shivs. The pcs make another faction.

- When the brawl starts, each group goes hostile with each other, but friendly within their own faction.
- When no one is in combat, the brawl should end, and reset back to "peaceful."
- When folks "wake up" from being knocked out, they should no longer be hostile.

The first of these was fairly straightforward... loop through the creatures in the bar, compare them to the other creatures, and if not in the same faction set them to hostile. The only real problem I had here was getting the nested loops working right. You can't use the "GetNext" family of functions for both loops if you are nesting. Once I realized that, and changed my loop indexing, it worked fine.

The second of these didn't take too long. I just added to the OnDeath event, and had it check to see if anyone in the area was still in combat. There were some problems with creatures remaining "in combat" after there were no enemies around. SurrenderToEnemies got around that problem, but caused others. In the end, I just delayed the check for a bit, to be sure that the "combat" state was clear.

The third one was the nightmare... I'll spare you the gruesome details. I just went back and forth, over and over, between two states. Either they would still be hostile when they woke up, or if they were neutral I couldn't get the brawl to start up again properly. This led to an *extensive* study of all the reputation commands, and I'm *still* not sure about all of them. However, finally, I realized that the source of all my problems was that I needed to make use of ClearPersonalReputation. Once I did that, everything fell into place, and I was able to get it working.

Three Points of Cleanup

Once the reputation problem was solved, I was really pretty happy with it. However, there were a few things that bugged me:

First, a mage could remove his weapons, and then fireball the patrons into cinders... but it was still “safe” combat as far as the scripts were concerned. This was easy to solve, I just added another user defined event, this one for OnSpellCastAt. If the spell was hostile, it ramped the combat up to deadly. I could pretty much cut and paste the OnDamaged code, and trim it down a bit. It took only a couple of minutes to fix.

The second one was something that I had been having nightmares about. Creatures tend to chase you across area transitions... so what happens if the pcs leave the bar during a brawl, and the patrons chase them outside?

I took the coward’s way out here. All I did was put a trigger in front of the exit. In the OnEnter script for the trigger, if it is not a pc, I just ClearAllActions. Simple, yet effective. It is still probably possible to have the npc chase the pc outside, given the right conditions... but in my testing, I haven’t seen it happen.

The final thing was to make it so that the pcs wouldn’t die in the safe fight either. This was simple, I just modified the OnDeath for the module, checked for the pc being in the area and the brawl being safe, and did the same trick I used for the npcs. I sent a message to the pc to inform him or her that it wasn’t “real death.” I dislike that there is no way to avoid having them show up as “Dead” under the cursor, but cest la vie.

Where to go from here?

There are many, many things that can be done to build on this idea. I never got around to making conversations for the patrons, and making alternate ways of starting the fight. Given that all I need to do is fire the area user defined event, though, that should be very straightforward.

Every rough bar needs bouncers. How are they going to behave in the fight?

What about the wait staff? Surely they won’t stand quietly by while the bar is thrown into chaos? Running and hiding seems appropriate.

We’ll add some of that stuff in with the next lesson. For now, let’s just pat ourselves on the back for a job well done.

The Final Scripts

Ok, here are the final versions that I used in the module, as well as instructions for building the area:

Build the tavern. I'll leave the details up to you, but I suggest I fairly large central area, a storeroom, a bedroom (for the innkeeper, doubles as a dressing room for the entertainers), and a couple other small rooms. Have only one exit outside.

Make a city area, and connect the two via a door transition.

Create the patrons inside the bar. I made 3 each of my three different factions, and made them level 3 fighters. Modify this as you wish, but be certain of a few things:

- Make each group a custom faction with parent of Defender, and make each faction 50 toward the others.
- Make sure the patrons have the “Weapon Proficiency: Simple” feat, so they can use their daggers.
- Do not give them any weapons, otherwise they will likely be too quick to use them.
- Give them all, no matter what faction, the tag SSPATRON
- Make sure the No Permanent Death box is checked.
- It is probably a good idea to make the appearances different for the three different groups.

Ok, now the scripts:

First, a library that I use in a couple of places:

NWScript:

```
// Reputation Library: fm_replib
//
// This library will eventually hold all of the custom
// functions I write for dealing with reputation. For the
// moment, only one has been needed.
//
// Written by: Celowin
// Last updated: 8/8/02
//
void NeutralReputation(object oTarget=OBJECT_SELF)
// This function neutrals out the reputation of oTarget
// with regards to every other creature in its area.
//
{
object oTargetArea=GetArea(oTarget);
object oCreature=GetFirstObjectInArea(oTargetArea);
while (oCreature != OBJECT_INVALID)
{
if (GetHitDice(oCreature)>0)
// Only affect creatures
```



```

    {
    ClearPersonalReputation(oCreature, oTarget);
    // Need to clear the personal reputation first, or everything
    // is flummoxed up.
    if (GetReputation(oTarget,oCreature)<=10)
    // Don't bother unless things are hostile.
    {
        AdjustReputation(oCreature,oTarget,50);
        AdjustReputation(oTarget,oCreature,0);
        // AdjustReputation seems to base the new reputation off
        // the better "direction" of the two. So this actually
        // sets both directions the same.
    }
    }
    oCreature=GetNextObjectInArea(oTargetArea);
}
}
}

```

The OnSpawn for the patrons:

NWScript:

```

// On Spawn Script: fm_sspatron_os
//
// This is the initialization script for the patrons of the
// Singing Spider Tavern. It is basically a stripped down
// version of the default BioWare script.
//
// Original version: Preston Watamaniuk
// Modified by: Celowin
// Last updated: 8/8/02
//
#include "NW_O2_CONINCLUDE"
#include "NW_I0_GENERIC"
void main()
{
SetSpawnInCondition(NW_FLAG_AMBIENT_ANIMATIONS);
//This will play Ambient Animations until the NPC sees an enemy or is
cleared.
//NOTE that these animations will play automatically for Encounter
Creatures.
// I don't believe I will need the other user defined events.
// However, it is better to leave them in and be safe, if I should
// change my mind at some later time.
//SetSpawnInCondition(NW_FLAG_HEARTBEAT_EVENT); //OPTIONAL
BEHAVIOR - Fire User Defined Event 1001
//SetSpawnInCondition(NW_FLAG_PERCIEVE_EVENT); //OPTIONAL
BEHAVIOR - Fire User Defined Event 1002
//SetSpawnInCondition(NW_FLAG_ATTACK_EVENT); //OPTIONAL
BEHAVIOR - Fire User Defined Event 1005
SetSpawnInCondition(NW_FLAG_DAMAGED_EVENT); //OPTIONAL
BEHAVIOR - Fire User Defined Event 1006
//SetSpawnInCondition(NW_FLAG_DISTURBED_EVENT); //OPTIONAL
BEHAVIOR - Fire User Defined Event 1008
SetSpawnInCondition(NW_FLAG_END_COMBAT_ROUND_EVENT); //OPTIONAL
BEHAVIOR - Fire User Defined Event 1003
//SetSpawnInCondition(NW_FLAG_ON_DIALOGUE_EVENT); //OPTIONAL
BEHAVIOR - Fire User Defined Event 1004
SetSpawnInCondition(NW_FLAG_DEATH_EVENT); //OPTIONAL
BEHAVIOR - Fire User Defined Event 1007
// I have no idea why this line isn't in the original script.
// Oh well, easy enough to add....
SetSpawnInCondition(NW_FLAG_SPELL_CAST_AT_EVENT); //OPTIONAL
BEHAVIOR - Fire User Defined Event 1011

```

```

// DEFAULT GENERIC BEHAVIOR (DO NOT TOUCH)
*****
SetListeningPatterns(); // Goes through and sets up which shouts
the NPC will listen to.
WalkWayPoints(); // Optional Parameter: void
WalkWayPoints(int nRun = FALSE, float fPause = 1.0)
// 1. Looks to see if any Way Points in the
module have the tag "WP_" + NPC TAG + "_0X", if so walk them
// 2. If the tag of the Way Point is
"POST_" + NPC TAG the creature will return this way point after
// combat.
GenerateNPCTreasure(); /* Use this to create a small amount of
treasure on the creature
}

```

The OnUserDefined for the patrons:

NWScript:

```

// User Defined Script: fm_sspatron_ud
//
// Sets up the special behaviors for a patron at the
// Singing Spider Tavern. Specifically, death is
// a fleeting thing. The npc will get up after a
// couple of minutes. This is supposed to simulate
// getting knocked out in a brawl.
//
// If the fight has turned deadly, death behaves
// pretty much as normal.
//
// Written by: Celowin
// Last Updated: 8/7/02
//
#include "fm_replib"
void main()
{
int nCalledBy = GetUserDefinedEventNumber();
// Even though only used in part of the switch,
// These need to be declared up top.
int nBrawl;
object oAttacker;
object oRWeapon;
object oLWeapon;
switch(nCalledBy)
{
case 1003: // On Combat Round End
if (GetLocalInt(OBJECT_SELF, "EQUIP")==1)
{
// If the fight just turned deadly, need to equip a weapon
// Create a dagger first
oRWeapon = CreateItemOnObject("nw_wswdg001",OBJECT_SELF);
// Equip the dagger
ClearAllActions();
ActionEquipItem(oRWeapon, INVENTORY_SLOT_RIGHTHAND);
// Remind the npc that he has equipped already
SetLocalInt(OBJECT_SELF, "EQUIP",0);
}
}
}

```

```

    }
    break;
case 1006: // On Damaged
    // If the brawl level is 1 (peaceful), immediately set it
    // up to 2 (safe brawl).
    nBrawl = GetLocalInt(GetArea(OBJECT_SELF), "BRAWL_LEVEL");
    if (nBrawl == 1)
    {
        nBrawl = 2;
        SetLocalInt(GetArea(OBJECT_SELF), "BRAWL_LEVEL", 2);
        SignalEvent(GetArea(OBJECT_SELF), EventUserDefined(3000));
    }
    // Next, if the attacker wielded a weapon, set the brawl
    // state up to 3 (deadly). Note that torches and shields
    // are considered out of place in a brawl, and will set it
    // off.
    if (nBrawl == 2)
    {
        oAttacker = GetLastAttacker();
        oRWeapon = GetItemInSlot(INVENTORY_SLOT_RIGHTHAND,
oAttacker);
        oLWeapon = GetItemInSlot(INVENTORY_SLOT_LEFTHAND,
oAttacker);
        if (GetIsObjectValid(oRWeapon) ||
GetIsObjectValid(oLWeapon))
        {
            SetLocalInt(GetArea(OBJECT_SELF), "BRAWL_LEVEL", 3);
            SignalEvent(GetArea(OBJECT_SELF), EventUserDefined(3050));
        }
    }
    break;
case 1007: // On Death
    if (GetLocalInt(GetArea(OBJECT_SELF), "BRAWL_LEVEL") == 2)
    // Check if the brawl is "safe."
    {
        // First, make sure the npc doesn't self destruct, and
        // can't be bothered during his or her "nap."
        SetIsDestroyable(FALSE, TRUE, FALSE);
        // A bit annoying, but this makes it so the npc won't be
        // hostile when waking up.
        SurrenderToEnemies();
        // After 2 minutes, res the npc.
        effect eRes = EffectResurrection();

        DelayCommand(120.0, ApplyEffectToObject(DURATION_TYPE_PERMANENT,
eRes, OBJECT_SELF));
        // Immediately after, heal the npc to half health.
        int nHealAmount = FloatToInt(
IntToFloat(GetMaxHitPoints())/2);
        effect eHeal = EffectHeal(nHealAmount);

        DelayCommand(120.1, ApplyEffectToObject(DURATION_TYPE_PERMANENT,
eHeal, OBJECT_SELF));
        // Finally, neutral out reputations
        DelayCommand(120.2, NeutralReputation());
    }
    else

```

```

        SetIsDestroyable(TRUE); // Otherwise, "real" death
        DelayCommand(15.0, SignalEvent(GetArea(OBJECT_SELF),
EventUserDefined(3100)));
        break;
    case 1011: // On Spell Cast At
        if (GetLastSpellHarmful()) // Do nothing if it was a nice
spell
        {
            // If the brawl level is 1 (peaceful), set up the
            // proper hostilities first.
            nBrawl = GetLocalInt(GetArea(OBJECT_SELF), "BRAWL_LEVEL");
            if (nBrawl == 1)
                SignalEvent(GetArea(OBJECT_SELF), EventUserDefined(3000));
            // Next, set the brawl to deadly.
            SetLocalInt(GetArea(OBJECT_SELF), "BRAWL_LEVEL", 3);
            SignalEvent(GetArea(OBJECT_SELF), EventUserDefined(3050));
        }
        break;
    }
}

```

The OnDeath script for the module:

NWScript:

```

// On Player Death Script: fm_onddeath
// Modified from NW_O0_DEATH.NSS
//
// Original Script by Brent Knowles
// Modification by Celowin
// Last Updated: 7/29/02
//
// For the most part, this script uses the default behavior.
// However, if the player is "killed" in the Singing Spider
// Tavern during a "safe" brawl, the player is given a
// message that they are merely unconscious, and will be
// woken up in 2 minutes.
//
#include "fm_replib"
void Raise(object oPlayer)
{
    effect eVisual = EffectVisualeffect(VFX_IMP_RESTORATION);
    effect eBad = GetFirstEffect(oPlayer);

    ApplyEffectToObject(DURATION_TYPE_INSTANT, EffectResurrection(), oP
layer);

    ApplyEffectToObject(DURATION_TYPE_INSTANT, EffectHeal(GetMaxHitPoi
nts(oPlayer)), oPlayer);
    //Search for negative effects
    while(GetIsEffectValid(eBad))
    {
        if (GetEffectType(eBad) ==
EFFECT_TYPE_ABILITY_DECREASE ||

```

```

        GetEffectType(eBad) == EFFECT_TYPE_AC_DECREASE ||
        GetEffectType(eBad) ==
EFFECT_TYPE_ATTACK_DECREASE ||
        GetEffectType(eBad) ==
EFFECT_TYPE_DAMAGE_DECREASE ||
        GetEffectType(eBad) ==
EFFECT_TYPE_DAMAGE_IMMUNITY_DECREASE ||
        GetEffectType(eBad) ==
EFFECT_TYPE_SAVING_THROW_DECREASE ||
        GetEffectType(eBad) ==
EFFECT_TYPE_SPELL_RESISTANCE_DECREASE ||
        GetEffectType(eBad) == EFFECT_TYPE_SKILL_DECREASE
||
        GetEffectType(eBad) == EFFECT_TYPE_BLINDNESS ||
        GetEffectType(eBad) == EFFECT_TYPE_DEAF ||
        GetEffectType(eBad) == EFFECT_TYPE_PARALYZE ||
        GetEffectType(eBad) == EFFECT_TYPE_NEGATIVELEVEL)
    {
        //Remove effect if it is negative.
        RemoveEffect(oPlayer, eBad);
    }
    eBad = GetNextEffect(oPlayer);
}
//Fire cast spell at event for the specified target
SignalEvent(oPlayer, EventSpellCastAt(OBJECT_SELF,
SPELL_RESTORATION, FALSE));
ApplyEffectToObject(DURATION_TYPE_INSTANT, eVisual,
oPlayer);
}
void main()
{
    object oPlayer = GetLastPlayerDied();
    // Find where the player died.
    string sArea = GetTag(GetArea(oPlayer));
    // If the player "died" in the brawl, deal with it.
    if (sArea == "SingingSpiderTavern" &&
(GetLocalInt(GetArea(oPlayer), "BRAWL_LEVEL")==2) )
    {
        SendMessageToPC(oPlayer, "You have been battered into
unconsciousness, and will awaken shortly.");
        FloatingTextStringOnCreature("(Unconscious)", oPlayer,
FALSE);
        effect eRes = EffectResurrection();
        int nMax = GetLocalInt(GetArea(oPlayer), GetName(oPlayer));
        int nHeal = FloatToInt( IntToFloat(nMax)/2 );
        effect eHeal = EffectHeal(nHeal);
        DelayCommand(120.0,
ApplyEffectToObject(DURATION_TYPE_INSTANT, eRes, oPlayer));
        DelayCommand(120.1,
ApplyEffectToObject(DURATION_TYPE_INSTANT, eHeal, oPlayer));
        DelayCommand(120.2, NeutralReputation(oPlayer));
    }
    else // Do the standard stuff
    {
        // * make friendly to Each of the 3 common factions
        AssignCommand(oPlayer, ClearAllActions());
        // * Note: waiting for Sophia to make

```

```

SetStandardFactionReputation to clear all personal reputation
    if (GetStandardFactionReputation(STANDARD_FACTION_COMMONER,
oPlayer) <= 10)
    {
        SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // *
Player bad
        SetStandardFactionReputation(STANDARD_FACTION_COMMONER,
80, oPlayer);
    }
    if (GetStandardFactionReputation(STANDARD_FACTION_MERCHANT,
oPlayer) <= 10)
    {
        SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // *
Player bad
        SetStandardFactionReputation(STANDARD_FACTION_MERCHANT,
80, oPlayer);
    }
    if (GetStandardFactionReputation(STANDARD_FACTION_DEFENDER,
oPlayer) <= 10)
    {
        SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // *
Player bad
        SetStandardFactionReputation(STANDARD_FACTION_DEFENDER,
80, oPlayer);
    }
    DelayCommand(2.5,
PopUpGUIPanel(oPlayer, GUI_PANEL_PLAYER_DEATH));
}
}

```

The OnEnter script for the tavern:

NWScript:

```

// On Enter Script:  fm_sspider_en
// This is the script for the area The Singing
// Spider Tavern.
//
// It does two things:
// 1. Initializes the brawl state to peaceful, if it
//    hasn't been set.
// 2. Stores the pc's current hit points on enter.
//
// Written by:  Celowin
// Last modified: 8/7/02
//
void main()
{
// Get the current brawl status.
int nBrawl = GetLocalInt(OBJECT_SELF, "BRAWL_LEVEL");
// If the status hasn't been set, set it to 1 (peaceful).
if (nBrawl == 0)
    SetLocalInt(OBJECT_SELF, "BRAWL_LEVEL", 1);
// Get the entering object, check if it is a pc, and if so,
// store the current hp.
object oPC = GetEnteringObject();
if (GetIsPC(oPC))
    SetLocalInt(OBJECT_SELF, GetName(oPC),
GetCurrentHitPoints(oPC));
}

```

```
}
```

The OnUserDefined script for the area:

NWScript:

```
// User Defined Event: fm_sspider_ud
// This script defines all the major events for the
// Singing Spider Tavern.
// Most of the events deal with the bar brawl that
// happens there.
//
// Written by: Celowin
// Last modified: 8/7/02
#include "nw_i0_generic"
void main()
{
int nCalledBy = GetUserDefinedEventNumber();
// Declarations up top, out of the switch.
object oCreature;
object oCreature2;
string sTag;
int nMax;
int nCurrent;
int nHealAmount;
effect eHeal;
int nIndex;
int bTest;
switch (nCalledBy)
{
case 3000: // The brawl just started
// Loop through, set the different groups hostile to each
// other and the pcs.
oCreature = GetFirstObjectInArea();
while (oCreature != OBJECT_INVALID)
{
if ( GetIsPC(oCreature) || (GetTag(oCreature) ==
"SSPATRON") )
{
nIndex = 0;
oCreature2 =
GetNearestObject(OBJECT_TYPE_CREATURE,oCreature,nIndex);
while (oCreature2 != OBJECT_INVALID)
{
if ( GetIsPC(oCreature2) || (GetTag(oCreature2) ==
"SSPATRON") )
{
if (!GetFactionEqual(oCreature,oCreature2))
AdjustReputation(oCreature,oCreature2,-100);
}
nIndex++;
oCreature2 =
GetNearestObject(OBJECT_TYPE_CREATURE,oCreature,nIndex);
}
}
oCreature = GetNextObjectInArea();
```

```

    }
    oCreature = GetFirstObjectInArea();
    // Now that they all hate each other, hhave them start
    fighting.
    while (oCreature != OBJECT_INVALID)
    {
        if (GetTag(oCreature) == "SSPATRON")
            AssignCommand(oCreature, DetermineCombatRound());
        oCreature = GetNextObjectInArea();
    }
    break;
case 3050: // The brawl just turned deadly.
    oCreature = GetFirstObjectInArea();
    while (oCreature != OBJECT_INVALID)
    {
        if (GetObjectType(oCreature) == OBJECT_TYPE_CREATURE)
        {
            if (!GetIsPC(oCreature))
            {
                sTag = GetTag(oCreature);
                if (sTag == "SSPATRON")
                {
                    // Tell the Creature to equip a weapon at the next
                    chance.

                    SetLocalInt(oCreature, "EQUIP", 1);
                    // Calculate the healing amount
                    nMax=GetMaxHitPoints(oCreature);
                    nCurrent=GetCurrentHitPoints(oCreature);
                    nHealAmount = FloatToInt( IntToFloat(nMax-nCurrent) /
                    2 );

                    // Heal the "fake" damage
                    eHeal = EffectHeal(nHealAmount);
                    ApplyEffectToObject(DURATION_TYPE_INSTANT, eHeal,
                    oCreature);
                }
            }
            else
            {
                // For pcs, healing but nothing else
                // Heals half the difference between current and what
                the

                // pc had when entering the tavern
                nMax = GetLocalInt(OBJECT_SELF, GetName(oCreature));
                nCurrent = GetCurrentHitPoints(oCreature);
                nHealAmount = FloatToInt( IntToFloat(nMax-nCurrent) / 2
                );

                // Apply the heal
                eHeal = EffectHeal(nHealAmount);
                ApplyEffectToObject(DURATION_TYPE_INSTANT, eHeal,
                oCreature);
            }
        }
        oCreature = GetNextObjectInArea();
    }
    break;
case 3100: // Check for end of brawl
    oCreature = GetFirstObjectInArea();

```



```

    bTest=FALSE;
    while (oCreature != OBJECT_INVALID)
    {
        // Check... are there any creatures in combat?
        if ((GetHitDice(oCreature)>0)&& GetIsInCombat(oCreature))
            bTest=TRUE;
        oCreature=GetNextObjectInArea();
    }
    if (bTest==FALSE) // If not, end the brawl
        SetLocalInt(OBJECT_SELF,"BRAWL_LEVEL",1);
    break;
}
}

```

Finally, paint a little trigger in front of the exit from the tavern, and attach the following to the OnEnter:

NWScript:

```

// On Enter Script: fm_ssdoortrig
//
// This script attempts to keep npcs from chasing
// pcs out of the tavern.
//
void main()
{
    object oPatron=GetEnteringObject();
    if (!GetIsPC(oPatron))
        AssignCommand(oPatron, ClearAllActions());
}

```

Conclusion

Whew! At any rate, have fun with it, and I hope that you learned something about the *process* of writing a complicated script. There is a lot of give and take, a lot of back and forth. We'll be doing a lot more with this in future lessons, so hold on to your hats. 😊

Also, feel free to speak up with extensions to this, or bugs that I might have missed. I know that the general procedure works, but there are always little things that get by. I've tried to fix as many as I can, but I'm sure there is some oddball thing that I have missed.

Finally, I want feedback on the new format. It is always tough to try to explain the "process" of doing something. I've done my best here, but if you have suggestions on what I could do to make it better, speak up.

Huntsmans Guide To Henchmen (Part 1) - SOU / HOTU Style

This tutorial aims to detail the points required to make your own custom henchmen in the SOU / HOTU style. Follow these steps to get yourself your basic template, a standard fighter with a normal package that will only level up in one class.

Creating The Basic Henchman.

1. Click on the word 'Wizards' at the top of your toolset and then click on the option 'Creature Wizard'.
2. On the wizard that opens click the 'NEXT' button at the bottom to move to the next screen where you can pick the racial type of henchman/creature you are going to create. Click on 'Human' in the list and then click the 'NEXT' button at the bottom.
3. You will notice on the next screen that you need to choose the class and level of the character. In this case a fighter of level one is already selected in the box to the right, and as this is what we are making clicking on the 'NEXT' button at the bottom.
4. On the next screen pick the gender of your henchperson, and choose a portrait for them. When you're done click the 'NEXT' button.
5. This screen allows you to choose the faction that the henchperson will be a member of before they are hired. Select 'MERCHANT' in the list and click the NEXT button again.
6. Give your henchman a name and click 'NEXT'.
7. This screen allows you to decide where on the custom creature palette the blueprint for this character will be located. Whilst you can of course place them anywhere, it is often a good idea to place your main characters together, for example in the 'Special : Custom 1' section. When you have finished click 'NEXT'. You will then be shown all the stats for your henchperson, check them and if they are ok click 'NEXT' again.
8. On the next screen click the box to 'Launch the creatures properties' and click the 'FINISH' button.
9. Click the tab marked 'SCRIPTS' and at the bottom click on the 'Load Script Set' button, and in the window that opens click on the file called 'set_xp1_henchmen.ini' and click the 'OPEN' button. It's possible the file's called something else on your machine, but look for one relating to henchmen.

10. You need a conversation for the henchman, since they are usually quite lengthy you can download one from [here](#). Unzip the hench_convoy.erf file and place it in your Neverwinter Nights erf directory. Then click on 'FILE' on the menu bar and select 'IMPORT' from the menu. In the window that opens, select 'hench_convoy.erf' and click 'OPEN'.

11. Click on the 'Basic' tab of your henchman's properties and look at the bottom for the 'Edit' button. Don't push that, but push the one to the left of it that looks like it has '...' at the bottom of it. Look for the file called 'hench_convoy' and select it and click ok. You should see the name of the file in the conversation window now. You can click the 'Edit' button now if you wish to edit the conversation, but you might be better off waiting until you've read the tutorial on 'One liners and interjections'.

12. Click on the 'Classes' tab of his properties and look at the 'Default Package for Autolevelup' window. Click the little arrow to the right and choose a package type for the henchman. It doesn't matter what you choose in this case.

You have now finished the basic henchman who will level up as a fighter, and will join you when you talk to him (due to the conversation file you added) and will react as the HotU henchmen do.

Multiclass Henchmen.

These are done exactly the same as you have done for the single class above, except that you need to make a change to the 'X0_I0_HENCHMAN' include file. (Note. If you imported the conversation file from above then you will already have a copy of 'X0_I0_HENCHMAN' to look at, edited to allow for henchmen with tags 'garey', 'olisha' and 'hench_elvalith'.) Here is what you do.

1. Click on the word 'Tools' at the top of the toolset and then click on the option for 'Script Editor'.
2. With the script editor open click on the option at the top to 'Open an existing file' (hold the mouse over the top row of buttons to see what they are).
3. Paste in the script name 'X0_I0_HENCHMAN' to the box marked 'Resource name' and make sure to check the box marked 'All resources' at the left hand side. Then click the 'Open' button.
4. If you look at around line 1364 you will see an entry that looks like this (it might be a slightly different line number depending on if you have HotU or SoU installed.).

```
else if (sTag == "x0_hen_dor") { LevelHenchmanUpTo(oAssociate,
nLevel, CLASS_TYPE_CLERIC, 20); }
```

5. Now this may look a little complex to some, but to be honest it's not and you don't need to fully understand it to actually use it. Look at the Tag of your henchman in your henchman's properties, and lets for this example say that it's 'Henchman_Bob'. You need to make your own entry to the file that is similar to the above, but using your henchman's tag, so yours would look like.

```
else if (sTag == "henchman_bob") { LevelHenchmanUpTo(oAssociate,
nLevel, CLASS_TYPE_CLERIC, 20); }
```

Just add your entry and save the script, but don't change the name of the script at all. (NOTE you must use lower case for the tag in the script even if your tag has capitol letters in it.)

6. The CLASS_TYPE_CLERIC can be changed to whatever you want the second class of your henchman to be. You can see what choices there are by looking at the Constants list by clicking on the 'CONSTANTS' button to the right of the script editor window and typing 'class_type' into the filter box at the top. So you might want the second class to be a rogue, and so would use CLASS_TYPE_ROGUE instead. The number 20 at the end of the line is the maximum level you want the henchman to level up in the second class that you are setting with this function in the script.

7. Lastly to make sure the henchman levels up when you do during game play click on the word "Edit" at the top of your toolset, and choose the option "Module Properties", and click on the "Events" tab. In the script slot OnPlayerLevelUp make sure the script name "x1_playerlevelup" is added there.

Controlling the amount of henchmen you can have.

If you want to control how many henchmen your PC can take with him you will need to make sure that the script that makes the henchman join (script called hire_henchman in my example file) uses the function AddHenchman(); rather than HireHenchman(); and then put the following line of script in the OnModuleLoad script slot of the modules property sheet.

```
SetMaxHenchmen(5);
```

The number in brackets is where you change it to however many henchmen you want the player to be able to hire. In this case 5.

If you do use more than two henchmen you will also have to make a change to the DoRespawn custom function that is listed in the x0_i0_henchman include at around line 995. This custom function controls the rehiring of the henchmen after they have died, and by default it uses HireHenchman(); which is no good as you would automatically fire a

henchman when the third one tried to rejoin. So replace HireHenchman() with AddHenchman(); and that will fix it.

Trouble Shooting

It is possible that using the scripts from SoU on the henchman that you might find weird things happen when the henchman dies. If you find this you might want to use the XP2 (HotU) scripts for OnDeath and OnSpellCastAt on the henchman. These scripts have the names x2_hen_death and x2_hen_spell and if you use these two scripts along with the ones you applied earlier in creating the henchman will cause you to have to use a resurrection spell or raise dead spell to revive the henchman.

Huntsmans Guide To Henchmen (Part 2) - One Liners, Interjections and module transfers.

This tutorial will be somewhat easier to follow if you already have the conversation file used in the 'Henchmen - SOU / HOTSU Style' tutorial. If you don't have it, you can download the file from [here](#).

A one liner, when it comes to a henchman, is when the henchman says something short requiring no response from the player. This usually occurs when the henchman crosses a trigger that you have laid down. Interjections are similar except that the henchman will actually start a conversation with the PC, rather than waiting for you to want to talk to him. It allows for a little more interaction from your henchmen companions.

One Liners - Non-Random.

In the conversation file you imported you will see a lot of single line NPC conversation lines, such as "We should be wary I think, all manner of nasties could hide in the shadows in here.", these are the one liners.

- 1) To get these to work you need to lay down one of the XP2 One_Liner, Non-Random triggers.
- 2) Open the properties of the trigger and go to the advanced tab. Change the Keytag to the number of the popup line that is to appear. So if it's the first line you want to appear put 1 in the keytag line.
- 3) In the conversation file you need to add a script to the text appears when tab of each Non-Random popup line. The script for line one is 'x0_d2_hen_line1' and for PopUp2 it needs 'x0_d2_hen_line2' and so on. Note the scripts only run up to 'x0_d2_hen_line52' though you could add more if you needed to.
- 4) Each one liner should have 'x0_d1_hen_noline' in the Actions Taken script box to clear the one liner.

One Liners - Random

Random One Liners work in a similar way to the non-random ones, but you don't need to change anything on the keytag of the trigger and use different scripts in the conversation. Before you can do any random one-liners or interjections though you will need to make a change to your henchman's OnSpawn script. If you look at their onSpawn script you will see code that looks like this.

```

else
if (sMyTag == "henchmans_tag")
{
SetNumberOfRandom("X2_L_RANDOMONELINERS", OBJECT_SELF, 5);
SetNumberOfRandom("X2_L_RANDOM_INTERJECTIONS", OBJECT_SELF, 2);
}

```

All you need to do is add in your own section of code like that, and place your henchman's tag where it says to in the above code. Change the number 5 and 2 to match the numbers of random one-liners and interjections in your conversation file for that henchman.

Now here is how to do the random one-liners.

- 1) Lay down an XP2 One-Liner, Random trigger.
- 2) In your conversation editor write your random one-liners and then use x2_d2_hen_rln001 script in the text appears when of the first line and then x2_d2_hen_rln002 for the second and so on.

That's all there is to the random one-liners.

Interjections - Non-Random.

Interjections work in pretty much the same way as the non-random popups above.

- 1) Lay down one of the XP2 Interjection, Non-Random triggers where you want the interjection to happen.
- 2) As with PopUps change the Key Tag to match the Interjection number you're using.
- 3) Whichever line of conversation you want to be used when the player walks over the trigger should have the script 'x0_d2_hen_inter' placed in the 'Text Appears When...' script box for the line. In the conversation file imported earlier this line is the one that goes "Wait , I would have a word with you if I may."
- 4) If the player chooses to continue the example conversation, the henchman has six possible conversation lines he might say. Each line has a script on the 'Text Appears When...' and 'Actions Taken' tab. The scripts on 'Text Appears When...' are in order and look like 'x0_d2_hen_inter1' or 'x0_d2_hen_inter2' etc and the script in 'Actions Taken' is ALWAYS x0_d1_hen_inter0 so as to reset the interjection and make sure it doesn't do it more than once. Note the script numbering runs up to 'x0_d2_hen_inter9' and then 'x0_d2_hen_inte10' to 'x0_d2_hen_inte50'.

Interjections - Random.

These work pretty much the same as the random popups do in that you don't need to change the keytags of the trigger.

- 1) Lay down an XP2 Interjection, Random trigger.
- 2) Add your random interjections below the non-random interjections you did above and then use the script x2_d2_hen_int001 for the text appears when tab of the first line and x2_d2_hen_int002 for line two, and so on for the rest of the random lines you do.

That is about all for the random interjections, but check out the conversation file in the example module for interjections at the bottom of this page to see how it all works in game and in a conversation file if you need to see it in action to help explain it.

Party Banter, or Party random Interjections

This is for when you have two or more henchmen with you and you want them to talk between themselves about something. It's fairly simple to do and similar to the random interjections we have already done, except that you will have to make your own scripts.

- 1) Lay down an XP2 PARTY Interjection, Random trigger where you want the interjection to happen.
- 2) Write in one of your henchman's conversation files what you want him to say to one of the other henchmen.
- 3) Write another OWNER line below the conversation you just wrote with the text "-----
- This line must be placed at the end of inter-party banter -----"
- 4) In the text appears when of the line you just wrote place the script x2_turnoffbanter and the script x2_hen_try_other goes in the Actions Taken script slot.
- 5) In the text appears when of the henchman's interjection you need to write a custom script that checks to see if the other henchman you want this henchman to talk to is in the party as well. An example of the type of script you need is below, just put your the tag of the henchman you're checking for where marked.
Also you will see that there is a 1 after the henchman tag place in the script. If you have more than one possible random line for the henchman you are trying a random banter with then that number is what you change. So for line one you use 1 and for the second possible banter you change that to 2.

```
#include "x2_inc_banter"  
int StartingConditional()  
{
```



```

if (TryBanterWith("henchman_tag_here", 1 ) == TRUE)
{
return TRUE;
}
return FALSE;
}

```

That is all you need for the banter, but if you need to see examples of how it works then download the below listed module that has examples of the henchmen doing all the above interjections and popups. SO you can see it in action in the module and then look at the conversation file in your toolset.

Only firing an interjections or Popups when part of a quest is done etc.

If you only want an interjection to fire if part of a quest has been done you would need to set an INT to the module first of all when whatever parameter has been met to allow the pop/interjection to fire. So you would need to have some idea of how to set INT's for this really, but if you do then all you need to do is use something like the below code in the OnEnter script of the XP2 popup/interjection trigger you are using.

```

void main()
{
object oEnter = GetEnteringObject();
if(GetLocalInt(GetModule(), "WHAT_EVER_NAME") == 1)
{
ExecuteScript("x0_evt_trigger", OBJECT_SELF);
}
}

```

SO what this does is check if a local int has been set to the module, and if it has then the script will fire when the henchman walks into the trigger and you will get the popup or interjection. If the INT hasn't been set then the script won't fire and no popup will occur.

That should about cover interjections and popups, as you can pretty much see how they need to be done from the popup and interjections example module, which can be found by [CLICKING HERE](#).

The zip file contains a module file that is HotU 1.61 compliant and an ERF file of the area for those that don't have HoTu but do have the 1.61 patch. You should be able to import it if you don't have HotU, but you may need to add the script to set the maxhenchmen to 2 in the OnModuleLoad (check lesson one on how).

Moving henchmen with you from one module to another.

To move a henchman with you from one module to another you need to set up a campaign database on both of the modules. Here is how you do that.

1) Place the below code in the OnModuleLoad script slot of BOTH of your modules.

```
void main()
{
    SetLocalString(GetModule(), "DB_GRAIL_WAR", "HenchMen");
}
```

2) Place the below code in the OnEnter script of a generic trigger just before you jump the PC to the new module.

```
#include "x0_i0_henchman"
void main()
{
    object oPC = GetEnteringObject();
    StoreCampaignHenchman(oPC);
}
```

3) Place the below code in the Area OnEnter script of the first area that the players enter.

```
#include "x0_i0_henchman"
void main()
{
    object oPC = GetEnteringObject();
    if(GetLocalInt(oPC, "LOAD_HENCHMAN") == 1)
        return;
    if(!GetIsPC(oPC))
        return;
    RetrieveCampaignHenchman(oPC);
    SetLocalInt(oPC, "LOAD_HENCHMAN", 1);
}
```

That's all there is to it if you're only going to be using a single henchman, but if you're intending the players to have more than one henchman you will need to do some code editing due to the fact bioware code only covered one henchman with their custom function, and while I will try to explain it clearly it's not as easy as previous bits.

If you look at around line 1096 of the x0_i0_henchman include file you will see a section of code that looks like this.

```
void StoreCampaignHenchman(object oPC)
{
    object oHench = GetHenchman(oPC, 1);
    if (!GetIsObjectValid(oHench)) {
        DBG_msg("No valid henchman to store");
        return;
    }
    DBG_msg("Storing henchman: " + GetTag(oHench));
    int ret = StoreCampaignDBObject(oPC, sStoredHenchmanVarname,
    oHench);
    if (!ret) {
        DBG_msg("Error attempting to store henchman");
    } else {
        DBG_msg("Henchman stored successfully");
    }
}
// Call this function when a PC enters a sequel module to restore
// the henchman (complete with inventory). The function
// StoreCampaignHenchman must have been called first, and both
// modules must use the same campaign db. (See notes in
x0_i0_campaign.)
//
// The restored henchman will automatically be re-hired and will
be
// created next to the PC.
//
// Any object in the module with the same tag as the henchman
will be
// destroyed (to remove duplicates).
void RetrieveCampaignHenchman(object oPC)
{
    location lLoc = GetLocation(oPC);
    object oHench = RetrieveCampaignDBObject(oPC,
    sStoredHenchmanVarname, lLoc);
    // Delete the henchman object from the db
    DelayCommand(0.5, DeleteCampaignDBVariable(oPC,
    sStoredHenchmanVarname));
    if (GetIsObjectValid(oHench)) {
        DelayCommand(0.5, HireHenchman(oPC, oHench));
        object oHenchDupe = GetNearestObjectByTag(GetTag(oHench),
        oHench);
        if (GetIsObjectValid(oHenchDupe) && oHenchDupe != oHench) {
            DestroyObject(oHenchDupe);
        }
    } else
    {
        DBG_msg("No valid henchman retrieved");
    }
}
```

You need to replace that section of code with the below code. This will then make it work for whatever your maxhenchman setting is. The max henchman allowed in a module is at line 75 of the x0_i0_henchman file, which is "int X2_NUMBER_HENCHMEN = 2;" just change the number 2 to whatever amount of henchmen you want to allow. So if your

module will be letting PC use 5 henchmen the line should be int
X2_NUMBER_HENCHMEN = 5;

```
void StoreCampaignHenchman(object oPC)
{
    object oHench;
    int iSlot, iMax = GetMaxHenchmen();
    for (iSlot = 1; iSlot <= iMax; iSlot++)
    {
        oHench = GetHenchman(oPC, iSlot);
        if (!GetIsObjectValid(oHench))
            DBG_msg("No valid henchman to store");
        else if (StoreCampaignDBObject(oPC, "Henchman" +
            IntToString(iSlot), oHench))
            DBG_msg("Henchman " + GetName(oHench) + " stored successfully");
        else
            DBG_msg("Error attempting to store henchman " + GetName(oHench));
    }
}
// Call this function when a PC enters a sequel module to restore
// the henchman (complete with inventory). The function
// StoreCampaignHenchman must have been called first, and both
// modules must use the same campaign db. (See notes in
x0_i0_campaign.)
//
// The restored henchman will automatically be re-hired and will
be
// created next to the PC.
//
// Any object in the module with the same tag as the henchman
will be
// destroyed (to remove duplicates).
void RetrieveCampaignHenchman(object oPC)
{
    location lLoc = GetLocation(oPC);
    object oHench, oDupe;
    int iSlot, iMax = GetMaxHenchmen();
    string sHench;
    for (iSlot = 1; iSlot <= iMax; iSlot++)
    {
        sHench = "Henchman" + IntToString(iSlot);
        oHench = RetrieveCampaignDBObject(oPC, sHench, lLoc);
        DelayCommand(0.5, DeleteCampaignDBVariable(oPC, sHench));
        if (GetIsObjectValid(oHench))
        {
            DelayCommand(0.5, HireHenchman(oPC, oHench));
            oDupe = GetNearestObjectByTag(GetTag(oHench), oHench);
            if (oDupe != OBJECT_INVALID) DestroyObject(oDupe);
        }
        else
            DBG_msg("No valid henchman retrieved");
    }
}
```

That should cover most things relating to using the SoU and HotU style henchmen, but I have little doubt I have forgotten a few things in which case you will have to catch up with myself or one of the other people on the bioware scripting forum to help you out, or work it out for yourself. :)

Trouble Shooting

Some of the subjects in this lesson mean changing the x0_i0_henchman file. Changing include files isn't always the best idea as it can lead to conflicts if Bioware update these include files in future expansions or patches. It can be best to save the changed include under a new name and then use that name for the include file of any scripts. So if you wanted you might resave the "x0_i0_henchman" script as "my_hench_include" and then in any script that you wanted to reference things from that script you would put the below above the void main of any script.

```
#include "my_hench_include"
```

Anyway, you don't have to do that, but it can be a good idea to do it that way to avoid possible conflicts. I hope this all was a help to people and if it was you're welcome to build a nice tasteful temple in my honour and pray to me as your god. :)

Introduction to struct

By Jassper

This tutorial assumes you have basic knowledge of scripting, functions, and include files. Structs are a more advanced topic and it is recommended you learn basic scripting before attempting to build a struct.

The Word Struct is short for "structure". By definition;

structure

1. Something built or erected; a building, bridge, framework, or other object that has been put together from many different parts
2. Orderly system of parts; a system or organization made up of interrelated parts functioning as an orderly whole

This is much how a struct is in coding, it is a data type (variables) that hold values of many other data types. It is a "structure" formed by many different variables.

Lets take a look at a struct that you are probably familiar with already, GetLocation(). This function returns the location of an object, that location actually contains 3 parts. The Area of the location, the vector of the location, and the facing of the object you are getting the location from. So GetLocation() contains a variable of data type object, a variable of data type vector, and a variable of data type float. But wait, a vector has 3 parts, x, y, and z. This is because a vector is also a struct, containing 3 variables of data type float.

To put it a simply way, a struct is nothing more than several variables placed into a single variable called a struct.

To make our own struct, we first need to declare the variables used within the struct just as we declare normal variables in side a void main. Or similarly, we need to draw the blue print to build the structure.

For this tutorial, we will construct a struct variable to hold some data about the PC. For this we will need the Players Name, The Players CD Key and the Players current Level. So lets layout our blueprint.

Open the script editor and comment out the void main () and the two {}'s. We won't need those as this will become our include scripts. I say, comment instead of delete because of a trick I'll show you later.

```
struct PCInfo
{
    string name;
    string key;
    int level;
};
//void main()
//{}
```

Save the file as "my_struct".

"Struct" is the data type, "PCInfo" is the variable name. The 3 variables inside the {} name, key, and level are the 3 bits of information that we are going to include in this single variable "PCInfo".

Notice the ";" after the last "}". This is important because remember, you are declaring a variable. So just as an int variable, int I = 1; you need a ; after a struct variable declaration as well.

Now that we have our blue print, we need a way to declare values for each variable inside the struct. For that we need a "structured function" or "Struct Funct" (say that 5 times fast). Basically, a structured function is nothing more than a custom function that places the data we need into the struct.

For those that have built your own functions, this will look somewhat familiar to you.

The syntax for a normal function is;

```
Return_value Function_Name [(parameters)]
```

A structured function syntax is;

```
Return_Value Variable_Name Function_Name[(parameters)]
```

For example, a normal function might be;

```
int My_Function(object oPC)
```

int is the Return_Value, My_Function is the Function_Name, and (object oPC) is the parameters.

We have already defined the variable in the script above, so when making a struct function we need to include the variable name so nwscrip knows which struct variable your talking about. Lets call the function GetData, so following the sysntax we would enter this;

```
struct PCInfo GetData(object oPC)
```

So struct is the Return_Value, PCInfo is the Variable_Name, GetData is the Function_Name and (object oPC) is the parameter.

In the same file we just created, add our structured function

```
struct PCInfo
{
    string name;
    string key;
    int level;
};
struct PCInfo GetData(object oPC)
{
}
//void main()
//{}
```

Before we can get any data we need, we must declare another variable name that will hold the information that we are going to return to the script, or a return value. Lets call

this variable "ValToReturn". So add this line to your struct function;

```
struct PCInfo ValToReturn;
```

Now we need to plug in our values but before we do that, I need to introduce you to the dot operator. The dot or period "." Is used to tell the struct that there is more information after the ValToReturn variable. Think of it as a whole number with a remainder. For example, 1.5 is telling us that the value is more than 1, but less than 2. Lets plug in our values, then I will explain more. Add the following these lines, now your function should look like this:

```
struct PCInfo
{
    string name;
    string key;
    int level;
};
struct PCInfo GetData(object oPC)
{
    struct PCInfo ValToReturn;
    ValToReturn.name = GetName(oPC);
    ValToReturn.key = GetPCPublicCDKey(oPC);
    ValToReturn.level = GetHitDice(oPC);
}

//void main()
//{}
```

Look at the variable ValToReturn as the "whole number" and the ".name" as the remainder. Notice that the name, key, level, are the same variables that we laid out in our struct PCInfo blue print. What this does is make the variable ValToReturn equal to the name, key, and level, all at the same time.

Now to show you my little trick, click the "save" button. You should not get any compile error, in fact you shouldn't get any message at all. So did that file save correctly? We won't know unless we compile it.

Now uncomment the void main() and the 2 {}'s then click the save button again. You should get the following error:

ERROR: NOT ALL CONTROL PATHS RETURN A VALUE

The void main() forces the struct function to compile as a normal script. This is a easy way to detect typos before you try to include the file in another script. The error produced is because we didn't enter a return for the function. Lets add that now,

```
struct PCInfo
{
    string name;
    string key;
    int level;
};
struct PCInfo GetData(object oPC)
```



```

    {
        struct PCInfo ValToReturn;
        ValToReturn.name = GetName(oPC);
        ValToReturn.key = GetPCPublicCDKey(oPC);
        ValToReturn.level = GetHitDice(oPC);
        return ValToReturn;
    }

void main()
{}

```

Resave the file and the error should go away. Don't forget to re-comment out the void main() part and save the file again. We now have our struct include file and we will use it in a script.

Open a new script window and above the void main() include the file we just created. We are also going to call the function in that file;

```

#include "my_struct"
void main()
{
    object oPC = GetEnteringObject();
    struct PCInfo sINFO = GetData(oPC);
}

```

Save it as "my_script". Everything at this point should compile without any errors. The struct PCInfo is telling the script which struct to use. In this case, the PCInfo structure. sINFO is the variable name we are calling this variable. Just like any other variable and that it is equal to our function GetData().

In our include file, we packed 3 variables into one called ValToReturn using the dot operator so to extract that information, we need to use the dot operator again, i.e., sINFO.name, sINFO.key and sINFO.level. Lets send this information to the Player so we can see it. Add the following so your script looks like this;

```

#include "my_struct"
void main()
{
    object oPC = GetEnteringObject();
    struct PCInfo sINFO = GetData(oPC);
    SendMessageToPC(oPC, sINFO.name);
    SendMessageToPC(oPC, sINFO.key);
    SendMessageToPC(oPC, IntToString(sINFO.level));
}

```

Remember that level is a int variable, so it must be converted to a string just as any other int variable before we send it as a message. Save the script and place it in the OnEnter of your starting area. Now test your module. In the chat window you should see your character name, your CD key, and your current level.

Note: the CD key will only be valid if you load the module up in nwserver first, then enter the game. Other wise you will only see the character name and your current level.

Congratulations! You just made your first struct.

II.SOU Misc Updates
(may be overridden by HOU Update)

Get2daString and Loops

By Georg Zoeller

There seems to be some confusion about Get2DAString and loops, so I think I clear them up a bit.

Get2DAString *should* not be used in loops, because it is a *slow* function. Looping through a 50 row 2DA can take several seconds, possibly very few rows (10) may be ok.

Get2DAString loads data dynamically into the game, meaning you can change the 2da while the game is running and calling the function will always return the current values from the 2da, which helps quite a lot if you are testing things, lets say a random treasure table.

There are several techniques you can use to avoid using 2DAs in loops, and I am sure the community will come up with some even more ingenious ideas 😊

Getting the maximum number of entries in the 2da:

If you want to use a 2DA to randomize things (i.e. a encounter table for a custom spawn system), you could use the row 0 field of a column to state how many rows your 2da has. Store the number onto a local integer (i.e. MaxRowsCol1 MaxRowsCol2). Run this function OnModuleLoad to initialize .

```
example my_2da
  Col1      Col2
0          3          1          <--- 3 entries in col0, 1 in col2
1          Text       Text
2          Text2      ****
3          Text3      ****
```

Whenever you want to spawn a random encounter, you can use Random(MaxRows) -1 to determine a line within the 2da from which to read.

Some more information:

- The function returns "" for "****" entries in the 2da
- The column headers are case sensitive, the 2da name is not
- You probably should prefix your own created 2das so you can distinguish them from the ones that ship with the game. Avoid using the prefix des_ we use it internally for design created 2das
- The function is very powerful to get information about the game rules that are not exposed via a scripting command.

- The function will return "" on any error (not existing 2da, row, column), so avoid giving empty rows a specific meaning in your code

Example: You want to know if a weapon does bludgeoning damage?

```
// return TRUE if weapon does bludgeoning damage
// Uses Get2DAStrng, avoid to use in loops!
int GetDoesBludgeoningDamage(object oWeapon)
{
    int nBaseType = GetBaseType(oWeapon);
    return
    (StringToInt(Get2DAStrng("baseitems", "WeaponType", nBaseType) ==
    2);
    // 2 = bludgeoning
}
```

... or want that direly missed GetWeight Function? .. check the TenthLbs row in Baseitems.2da

It is probably a good idea to state a warning about not using the function in loops in its comments, because it uses Get2DAStrng internally.

I am sure we will see some community created include files with the most useful functions pretty soon

- Maximum length for a 2da name is 16 chars. If you go beyond, the function will fail without warning

- If you suspect that you would need data read once from a 2da later, you can implement simple caching to speed up future access to the data

Example

monster ResRef from a 2da driven random encounter system:

```
my_encounters.2da
    MonsterResRef
0    3                <--- number of resrefs
1    nw_balor
2    nw_deer
3    nw_dog
```

Lets assume you built a random encounter system as described above.

```
// code sample without caching
int nRow = Random(nMaxRows)+1;
string sMonster =
Get2DAStrng("my_encounters", "MonsterResRef", nRow);
//Get a monster from the 2da
CreateObject(...); //spawn the monster
```

Now lets add simple caching to the same code to avoid calling Get2DAStrng :

```
int nRow = Random(nMaxRows)+1;
string sMonster;
// get the monster from the cache
sMonster = GetLocalString(GetModule(),"Monster" +
IntToString(nRowNumber));
if (sMonster == "")
{
    // cache miss
    sMonster = Get2DAStrng("my_encounters", "MonsterResRef", nRow);
    //cache the monster on the module
    SetLocalString(GetModule(),"Monster" + IntToString(nRow),
sMonster);
}
CreateObject(...);
```

This way Get2DAStrng gets only called the first time you spawn an a certain monster. Any subsequent call will be handled by the stored variable which is significantly faster.

Of course you pay for this optimization by forfeiting the ability to change 2DAs outside the game while the module is running, but you probably wouldn't do this for a random encounter system anyway 😊

Have fun tinkering around with this, Get2DAStrng is an extremely powerful tool

The SoU Treasure System

By Georg Zoeller

It is documented extensively in x0_i0_treasure.nss

Basically you just need to place down chests and put items in them. There are module and area specific chests as well as monster race specific chests.

You find these chests in the toolset palette, their tags start with x0_

You can examine the official SoU campaign to learn more about using them

excerpt:

```
///://////////////////////////////////////
///: X0_I0_TREASURE
/*
Include library for XP1 treasure generation system.
-----
MAIN CONCEPT:
The module builder creates some base containers and fills them
with instances of the treasure items that s/he wants to appear
randomly. The actual treasure containers then randomly pick from
the contents of these base containers to generate their own
contents.
-----
SETUP:
Create four module-wide base containers (for low, medium, high,
and unique treasure items) and place them anywhere in your
module. Make sure that they are in locations inaccessible to the
players, though!
Give these module-wide base containers these tags (or just use
the
blueprints):
    X0_MOD_TREASURE_LOW           - low-level treasure
    X0_MOD_TREASURE_MED           - medium-level treasure
    X0_MOD_TREASURE_HIGH          - high-level treasure
    X0_MOD_TREASURE_UNIQ         - unique treasure items
Fill the instances of these base containers with appropriate
treasure
for your module.
For any areas where you want finer control over the treasure,
simply create additional base containers (you don't need all four
-- any one that you skip will fall back to the module-wide
version)
and place them in the area. Give them these tags (same just
without
the "MOD"):
    X0_TREASURE_LOW              - low-level treasure
    X0_TREASURE_MED              - medium-level treasure
    X0_TREASURE_HIGH             - high-level treasure
```

X0_TREASURE_UNIQ - unique treasure items

For any treasure container, use one of the following scripts as BOTH the OnOpen/OnDeath handler:

Any Treasure: x0_o2_any{low,med,high,uniq}

Books (book, scroll): x0_o2_book{low,med,high,uniq}

Potions: x0_o2_potn{low,med,high,uniq}

Armor (armor, shield, helm, boots, etc):

x0_o2_arm{low,med,high,uniq}

Weapon: x0_o2_weap{low,med,high,uniq}

Gold: x0_o2_gold{low,med,high,uniq}

Others may also be added.

MONSTER/NPC TREASURE

If you would like to have special monster treasure (monster treasure defaults to low treasure otherwise), you can also add any of these base chests:

X0_TREASURE_MONSTER_<monster tag> - treasure found on monsters with the given tag. This will strip off any trailing digits, so for instance, NW_KOBOLD1, NW_KOBOLD2, NW_KOBOLD005 would all end up checking the chest

X0_TREASURE_MONSTER_NW_KOBOLD.

X0_TREASURE_MONSTER_<racialtype> - treasure found on monsters of the given racialtype. Ex:
X0_TREASURE_MONSTER_ELF
X0_TREASURE_MONSTER_UNDEAD
The spelling matches that used in the RACIALTYPE_* constants.

X0_TREASURE_MONSTER - generic treasure found on all monsters/NPCs in the module.

To use monster treasure, use the default OnSpawn script nw_c2_default9 or modify the OnSpawn script as follows:

- Replace #include "nw_o2_coninclude" with #include "x0_i0_treasure"
- Replace GenerateNPCTreasure(); with CTG_GenerateNPCTreasure();
- If you prefer to generate monster treasure from the general chests, you can also add a parameter to CTG_GenerateNPCTreasure(); to specify whether the treasure generated should be low/medium/high/uniq.

See the comments to that function for details.

DETAILS:

Each treasure container has a script that looks for the nearest object with the tag matching the type of

treasure.

When the treasure-generating scripts look for a container, they will take the nearest container in the local area, then fall back to the module-wide container. If no container can be found, they will fall back on the standard random treasure generation system.

The treasure generation system will randomly select from 1-3 items from whatever items are in the nearest container. Each item has a 50/50 chance of being replaced by a random amount of coin based on the standard treasure generation system.

IMPORTANT NOTE ON PROBABILITY:

Each item in the chest has an equal probability of being selected. To control the probability of any given item appearing, simply place multiple copies of it in the container. Because of the stacking rules, there is a slight modification to this. Ammunition/throwing weapons will have a probability equal to the number of stacks in the chest, since the max stack size is the default stack size. Potions, scrolls, and gems, however,

will be counted individually to determine the probability even if they are stacked, since their default stack size is 1.

To control the size of the stack that actually gets created, you should create a blueprint that has the appropriate stack size.

You can adjust the stack size of the instance you stuff into the base container to whatever you want afterwards; the blueprint's stack size will control what

There are scripts that will cause only specific types of items to be selected from the container of the specified level -- eg, only books/scrolls, or only weapons or armor, etc -- useful for bookcases or for weapon racks, & so on.

The unique treasure chest will only place each item in the chest once, will never place more than one item, and will never replace the item with coin.

III. HOU Misc Updates

HoU Variables

NWN:HotU has a selection of variables that can be set to override the default behavior/rules in the game.

Examples include:

- Activating/Deactivating the Use magic device skillroll for rogues
- Toggle if the WalkWaypoints command makes creatures cross area borders
- Enable/Disable the New Item creation feats
- Modify a creatures AI behavior (i.e. % of magic use, compassion, offense, etc)
- and a lot more

All these options are documented in a file called **x2_inc_switches.nss**. To see an example on how to configure them for your module, open the file **x2_mod_def_load.nss** which is automatically attached to the OnLoad slot of each new module created with Hordes of the Underdark. I will add more detailed information about how to write your own additions to the combat AI and how to modify the default AI behavior later in this thread.

Dynamic Item Properties

To get started with dynamic item properties, I suggest the file **x2_inc_itemprop.nss** which holds wrapper functions for most of the common tasks you will encounter when dealing with dynamic item property scripting. The file is reasonably well commented so an experienced script shouldn't have too much trouble to find out what's going on.

To see examples of how to add item properties to an item, I suggest the new HotU spellscripts for i.e. magic weapon (**x2_s0_magcweap.nss**), holy sword (**x2_s0_holyswrld.nss**), or flame weapons (**x2_s0_x2_s0_flmeweap.nss**)

You may want to have a special look at the function *IPSafeAddItemProperty* which is the universal wrapper function used throughout the HotU Spellscripts when it comes to adding itemproperties. It takes care of most problems that can arise (i.e. preventing a temporary item property from removing a permanent one, etc). I will post code examples about this later.

On Hit Cast Spell

With HotU, you can create your own OnHit spells for weapons and armor, in addition to the spells already available in the toolset (much like the CastSpell:Unique Power ability).

In order to add your own custom OnHitCastSpell to a weapon, you first need to add the OnHit: Cast Spell - Unique Power property to the weapon or armor in question. Then pen and modify the file **x2_s3_onhitcast.nss** in your module. The file is well

commented so you should have no trouble to figure out what to do. I will post some code examples about this topic later.

Item Event Routing

Any new module created with the Hordes of the Underdark toolset has a couple of default script attached to its events. These default script allow routing of item specific events into a single file, so you need to have only one script file that contains all the scripts for your items.

Note that this feature only works if you have the default xp2 module scripts on your module (or added the appropriate code to your custom scripts).

To get started, have a look at the file **x2_it_example.nss**, it should be well enough commented to get you started on that topic. I will add a short example on this topic later.

Post-SOU Doors – Old ones don't work

You will need to replace your secret doors!

- 1) Remove all "Hidden Wall Door Trigger"s
- 2) Place a trigger in front of the wall where you want the door to appear. Name it's tag whatever you like! (Note there is special trigger for this under "NEW TRIGGERS")
Also note there are three types of secret wall doors - Wood Stone and metal)

Example: SecretDoor1A

- 3) Now create a waypoint in the location you want the door to appear. Name it's tag with the prefix LOC_ follow by the tag name of the trigger you created.

Example LOC_SecretDoor1A

NOTE: Watch out! Make sure the arrow is pointing in the direction you want the PC to walk through. If the arrow is pointing towards you, the door will appear backwards and will not open.

- 4) Finally, Place another Waypoint in the spot where you want the PC to appear after going through the secret door. The tag name for this is prefixed by DST_ and then the tag name of the trigger.

Example: DST_SecretDoor 1A

Note: Do not place the DST_ waypoints to close to the wall. When doing back to back secret doors, a PC can get stuck against the door. A good inch away from the wall seems to be good.

5) Special Note: You can adjust the DC of detecting a door by changing the number in the KEY TAG space under the TRIGGERS properties.

Hope this helps people! I know I spent hours trying to get my secret doors to work right.

Script Caching

By Georg Zoeller

How to decide which scripts to cache?

When you try to decide which scripts to use with the new script caching feature (module->properties>cached scripts) you should know how it works:

- The function allows to cache compiled scripts (.ncs) into RAM, thus speeding up their execution time
- Since it caches compiled scripts, it makes no sense to cache include files, as they are not compiled
- How many scripts you should/can cache is dependent on your computers RAM. Please keep in mind that this is stored on the module, so if you create modules for the public, you should remember that many people's computers may have less RAM than you have at your computer
- It makes sense to cache the most used scripts in your campaign, so heartbeat scripts, spawn in scripts for often-used monsters, etc make sense to cache, while caching an OnRest event script may not yield any noticeable performance increase in your module.

Hope that helps

Fun With Petrify

(Syruso)

I was playing around in the scripts and I found two versions of a petrification effect function.

Look in x0_i0_petrify

One applied a stone effect and the other applied a wood effect.

From reading these - I would draw two conclusions:

1) The normal 'petrify' function does not apply a separate vfx and thus I would assume that the base petrify effect applies this visual effect automatically. (On the other hand,

The wooden look applies a barkskin vfx on top of the petrification effect).

2) We will soon be inundated with characters and creatures being turned to wood and stone.

You might be able to get back to the normal appearance of the creature by applying some other visual effect (like Barkskin) and then quickly dispelling it. Given how that type of thing usually works, odds are you will end up with the petrification in place but without the appearance. Give it a shot.

```
void main()
{
  object oPC = GetEnteringObject();
  object oLiz = GetObjectByTag("Liz");
  effect eStone = EffectPetrify();
  AssignCommand(oLiz, ActionPlayAnimation(ANIMATION_FIREFORGET_TAUNT
));
  DelayCommand(1.0, AssignCommand(oLiz, ApplyEffectToObject(DURATION_
TYPE_PERMANENT, eStone, oLiz)));
}
```

Adding Crafting Materials to Your Campaign

In order to allow your players to use their crafting skills, you need to make sure that crafting materials are available to them.

The easiest way to do that is to add the script x2_o2_dead.nss to the OnDeath event of destroyable placeables and Doors, like it is used in the Official HotU campaign. This will cause the placeable/door to drop some crafting materials if it is destroyed by a character skilled enough in crafting (craft armor or craft weapon 5+) to salvage the material.

The materials that are dropped by this script are defined in des_crft_drop.2da where the row index in that file corresponds to the appearance type of the placeable/door/creature calling the script. You do not need to distribute a customized version of this file if you make any changes to it (i.e. via hakpak), as the script using the file is server side only (this way you can hide your drop tables from your players).

The default death scripts will take care of dropping creature related material (i.e. hides, feathers or metal for certain types of golems).

How to add more dyekits

Horde of the Underdark comes with about 15 different colors for each of the 3 dyetypes (leather, cloth, metal). Of course this won't be enough for a lot of people, so here a quick guide on how to add a new dyekit.

1. Create a new misc. small item with the dyekit icon (or whatever), best by copying an existing dyekit

2. Add the appropriate Item Property: Cast Spell properties to it:

Example: Leather Dye - Add the following properties

Cast Spell: Dye Leather 1 (1 charge / use)

Cast Spell: Dye Leather 2 (1 charge / use)

3. Set the number of charges to the item (we use 5 charges for the official dyekits) and modify the cost to suit your module's economy

4. Here comes the somewhat tricky part: Defining the color. In order to define the color of your dyekit, you need to set its tag according to the following rules:

MyTagXX

where MyTag is any string you want up to 14 chars, and XX is a two digit number between 0 and 63 that represents the color the dyekit will have. To find out the number of a color, you need to open the armor color palette in the toolset (i.e. by editing an armor) and count on that palette from left to right, starting 00 in the first row until you reach the color you want to your dyekit to have.

Example: If you want your metal color dyekit (CastSpell: Dye Metal 1 & 2) to give red color, the tag of your new dyekit needs to end with the digits 24. (I'll eventually add a picture link here, maybe someone from the ccg can make one 😊)

5. Save your dyekit and put it into your module and it is ready to go. You might want to create a copy of the Dyekit merchant in the store's palette that has your custom dyekit in the selection.

So in the end, if you want a dyekit for every color, you will have to create 2x 64 items (minus the ones that already exist) that will have tags from MyTag00 to MyTag63.

If you are interested in the script that is used for the dyekits, have a look at `x2_s2_dyearmor.nss`.

In case you are wondering why I am stripping any temporary item properties from the dyed armor in that script: the CopyAndModifyItem (and CopyItem) scripting command causes the item properties on the armor to be detached from their creator (as the object id changes) and since the time for temporary effects and properties is tracked on the creator, existing temporary properties (i.e. from a magic vestment spell) would never go away from the dyed armor until the player logs out. Removing all temporary item properties prevents that from happening.

And for the lazy people...

Android79 created dyekits for every possible color and uploaded them as importable .erf file to the vault:

[Click Here](#)

HotU comes with Craft Wand, Brew Potion and Scribe Scrolls feats.

The actual crafting feat system is way too complicated to describe it with a few lines here, but to give you some head start:

If you want to modify the crafting code, you need to check the file x2_pc_craft.nss. It is executed at the start of any spellscript. Since it is a single script file, you do not need to recompile any additional scripts to do any changes to the crafting system.

The actual implementation for the crafting system is can be found in x2_inc_craft, its pretty well commented. There are scripting hooks we left in for the community to create things like material components for crafting in this file.

Also note that a couple of options in the crafting system (i.e. always 50 charges on any crafted wand) can be toggled using module switches as outlined in the first post of this thread.

Most data for the crafting system is taken from des_crft_spells.2da. This file allows a builder to disable certain spells for the crafting feats (i.e. timestop). As with all des_*.2da files, this file does not need to be present on the client.

Also des_crft Scrolls.2da holds a list of templates for the Scribe Scroll feats, pretty much straightforward.

Prestige Class Control Variables

On popular demand, the variables that you can set on a PC to prevent him from taking a certain xp2 prestige class. You need to set the variable to 1 to block access to that prestige class:

```
Weapon Master: X2_AllowWM  
Shifter: X2_AllowShiftr  
Pale Master: X2_AllowPalema  
Dwarven Defender: X1_AllowDwDef  
Dragon Disciple: X1_AllowDrDis  
Champion of Torm: X2_AllowDivcha
```

So basically, if you want to prevent people from taking a certain prestige class in your module, i.e. until they have taken finished a certain quest, all you need to do is to put this into the OnClientEnter script:

```
object oPC = GetEnteringObject();
// no weapon master levels yet ...
if (GetLevelByClass(oPC, CLASS_TYPE_WEAPON_MASTER) == 0)
{
    SetLocalInt(oPC, "X2_AllowWM", 1);
}
// no weapon master levels yet ...
if (GetLevelByClass(oPC, CLASS_TYPE_SHIFTER) == 0)
{
    SetLocalInt(oPC, "X2_AllowShiftr", 1);
}
// ... and so on
```

Later you just need to remove this variable if a player has done the quest you want him to do ... and he will be able to level up in that class.

New scripting functions spotlight

A spotlight on a few of the new scripting functions, just to give you a few ideas on what could be done. I plan on adding more functions to this list over time.

GetLineOfSightObject

```
// Returns whether or not there is a direct line of sight
// between the two objects. (Not blocked by any geometry).
//
// PLEASE NOTE: This is an expensive function and may
//              degrade performance if used frequently.
int LineOfSightObject( object oSource, object oTarget )
```

Comment: Pretty cool new function, but really slow, avoid using it in loops or frequently called scripts

EffectCutsceneImmobilize

```
// Returns an effect that when applied will paralyze the target's
// legs, rendering
// them unable to walk but otherwise unpenalized. This effect
// cannot be resisted.
effect EffectCutsceneImmobilize()
```

Comment: Very helpful to freeze a player into a spot while leaving him able to act (i.e. for conversations). Note that a character's Dex is set to 3 for the duration of this effect

EffectCutSceneGhost

```
// Creates a cutscene ghost effect, this will allow creatures
// to pathfind through other creatures without bumping into them
// for the duration of the effect.
effect EffectCutsceneGhost()
```

Comment: A creature affected by this effect will ignore other people personal space, walking right through them (example: gelatinous cube). Note that there is a variable you can set on a creature in the toolset to make them spawn with this effect permanently (check x2_inc_switches for details)

SetMaxHenchmen

```
// Sets the maximum number of henchmen  
void SetMaxHenchmen( int nNumHenchmen )
```

Comment: Pretty straightforward. If you want to allow players to have more than one henchmen in your module, you want to call this function in your module's OnLoad script.

GetMaxHenchmen

```
// Gets the maximum number of henchmen  
int GetMaxHenchmen( int nNumHenchmen )
```

Comment: Return the maximum number of allowed henchmen per player in the current module.

DayToNight

```
// Changes the current Day/Night cycle for this player to night  
// - oPlayer: which player to change the lighting for  
// - fTransitionTime: how long the transition should take  
void DayToNight(object oPlayer, float fTransitionTime=0.0f)
```

Comment: If you played HotU chapter 3, you know this. It will change the lighting settings for a single player, so he sees the area as night, while everyone else in the area will see it as day. This does only manipulate a clients visual perception, not the actual time on the server.

NightToDay

(see DayToNight)

ForceRest

```
// Gives this creature the benefits of a rest (restored  
hitpoints, spells, feats, etc..)  
ForceRest()
```

Comment: If you want to write your own resting system, this is your friend. You could also use it to make monsters rest when a player leaves for some nasty surprises.

An introduction to Tile Magic

By Georg Zoeller

(Sunjammer's excellent script/placeable package:

<http://nwn.bioware.com/forums/viewtopic.html?topic=373740&forum=47>

If you played chapter 3 of the official campaign, you probably asked yourself several times "how did they do this, they can't be changing the tilesets on the fly?!"

Without a spoiler, here is a short introduction to Tile Magic(tm):

The file you want to look at is "x2_inc_toollib.nss". Two of the functions in this file are dealing with changing tiles in a tileset (or better creating the illusion to do that).

To give you a short idea of what you can do with it, try this example:

Example 1:

1. Create a new module with a new forest area, 8x8 tiles
2. Create a pit in the middle of the forest area, maybe 3x3 tiles
3. Stick the following script your module:

```
#include "x2_inc_toollib"
void main()
{
    int nX = 8;
    int nY = 8;

    TLChangeAreaGroundTiles(GetArea(OBJECT_SELF), X2_TL_GROUNDTILE_WATER, nX, nY, -0.92f);
}
```

4. Start your module, enter the area, enable debugmode and run your script using the runscrip command

Example 2:

1. Create an City Interior area, 4x4 tiles
2. Paint town an Inn
3. Place some broken furniture placeables around
4. Save this script into your module:

```
#include "x2_inc_toollib"
void main()
{
    int nX = 4;
    int nY = 4;

    TLChangeAreaGroundTiles(GetArea(OBJECT_SELF), X2_TL_GROUNDTILE_WATER,
```

```

ER, nX, nY, 0.3f);
}

```

5. Run the script via the console's runscript command.

6. Woa, what a bad place to live

The z-offset is dependent on the tileset. If you use any other tileset than the forest tileset, you need to play around with the z-offset values in the function!

Of course this is just one of the many things you can do using the technique we used. You've probably seen many other applications in the Hordes of the Underdark official campaign, especially Chapter 2 and 3. If not ... what are you waiting for, play the game!

Apply/Remove Tilemagic from Specific Tiles

By satchmogold

Here's an include file I created that allows you to add and remove tile magic to specific tiles, instead of just effecting a range of the area. I posted this in another thread, but thought it would be easier to find with its own topic. The include file has two functions, TLChangeSpecificAreaGroundTile and TLResetSpecificAreaGroundTile. These functions have the same parameters as the original tile magic functions. The only difference is that the row and column parameters specify the one tile to be effected, instead of a range of the area to effect.

```

//:////////////////////
//: satchmogold's VFX Tool library
//: sc_inc_mytmagic
//:////////////////////
/* ***** INTERFACE *****
*****
const int X2_TL_GROUNDTILE_ICE = 426;
const int X2_TL_GROUNDTILE_WATER = 401;
const int X2_TL_GROUNDTILE_GRASS = 402;
const int X2_TL_GROUNDTILE_LAVA_FOUNTAIN = 349; // ugly
const int X2_TL_GROUNDTILE_LAVA = 350;
const int X2_TL_GROUNDTILE_CAVEFLOOR = 406;
const int X2_TL_GROUNDTILE_SEWER_WATER = 461;
// change the type of the ground or (by default) sub ground tiles
(i.e. water) to the specified type
// Valid values are
// const int X2_TL_GROUNDTILE_ICE = 426;
// const int X2_TL_GROUNDTILE_WATER = 401;
// const int X2_TL_GROUNDTILE_GRASS = 402;
// const int X2_TL_GROUNDTILE_LAVA_FOUNTAIN = 349; // ugly
// const int X2_TL_GROUNDTILE_LAVA = 350;
// const int X2_TL_GROUNDTILE_CAVEFLOOR = 406;
// const int X2_TL_GROUNDTILE_SEWER_WATER = 461;
/* ***** IMPLEMENTATION *****
void TLResetSpecificAreaGroundTile(object oArea, int nColumn, int
nRow)
{

```

```

    object oTile = GetFirstObjectInArea(oArea);
    float x = 5.0 + ( 10.0 * nRow );
    float y = 5.0 + ( 10.0 * nColumn );
    vector vPos;
    while (GetIsObjectValid(oTile))
    {
        if (GetObjectType(oTile) == OBJECT_TYPE_PLACEABLE &&
GetTag(oTile) == "x2_tmp_tile")
        {
            vPos = GetPosition( oTile );
            if ( vPos.x == x && vPos.y == y )
            {
                SetPlotFlag(oTile,FALSE);
                DestroyObject (oTile);
            }
        }
        oTile = GetNextObjectInArea(oArea);
    }
}
void TLChangeSpecificAreaGroundTile(object oArea, int
nGroundTileConst, int nColumn, int nRow, float fZOffset = -0.4f )
{
    object oTile;
    vector vPos;
    vPos.x = 5.0 + ( 10.0 * nRow );
    vPos.y = 5.0 + ( 10.0 * nColumn );
    vPos.z = fZOffset;
    float fFace = 0.0;
    location lLoc = Location(oArea, vPos, fFace);
    oTile = CreateObject(OBJECT_TYPE_PLACEABLE, "plc_invisobj",
lLoc,FALSE, "x2_tmp_tile");
    SetPlotFlag(oTile,TRUE);
    ApplyEffectToObject(DURATION_TYPE_PERMANENT,
EffectVisualEffect(nGroundTileConst), oTile);
}

```

Sunjammer's Take:

I took a slightly different approach so that TileMagic could be used for a rectangular range of one or more tiles.

```

// -----
// sj_tilemagic_i
// -----
/*
    More flexible "tilemagic" commands.
*/
// -----
/*
    12/03 - v1.00 - Sunjammer
    - created
*/
// -----
#include "x2_inc_toollib"

//
*****
// * PROTOTYPES *

```

```

//
*****
// Creates a layer of ground or sub-ground VFX tiles of the
specified type
// within the specified range. Note the first column and row in
an area is 0,0.
// Also the various ground and sub-ground tiles have individual
z-axis offsets.
// - oArea:          area affected
// - nGroundTile:    any X2_TL_GROUNDTILE_* constant
// - nColumn:        column index of bottom left corner
// - nRow:           row index of bottom left corner
// - nColumns:       number of columns to be filled, must be
greater than 0
// - nRows:          number of rows to be filled, must be greater
than 0
// - fZOffset:       z-axis offset (height) of tiles
void TLChangeAreaGroundTileRange(object oArea, int nGroundTile,
int nColumn, int nRow, int nColumns=1, int nRows=1, float
fZOffset = -0.4f );
// Resets any ground or sub-ground VFX tiles previously created
within the
// specified range.
// - oArea:          area affected
// - nColumn:        column index of bottom left corner
// - nRow:           row index of bottom left corner
// - nColumns:       number of columns to be reset, must be
greater than 0
// - nRows:          number of rows to be reset, must be greater
than 0
void TLResetAreaGroundTileRange(object oArea, int nColumn, int
nRow, int nColumns=1, int nRows=1);

// *****
// * FUNCTIONS *
//
*****
void TLChangeAreaGroundTileRange(object oArea, int nGroundTile,
int nColumn, int nRow, int nColumns=1, int nRows=1, float
fZOffset = -0.4f )
{
    object oTile;
    location lLoc;
    // sanity check
    if(nColumns < 1 || nRows < 1) return;
    // find center of first bottom left tile in range
    float fX = 10.0 * nColumn + 5.0;
    float fY = 10.0 * nRow + 5.0;
    vector vPos = Vector(fX, fY, fZOffset);
    // fill range
    int i, j;
    for (i = nColumn ; i < nColumn + nColumns; i++)
    {
        for (j = nRow; j < nRow + nRows; j++)
        {
            // create VFX tile
            lLoc = Location(oArea, vPos, 0.0);

```

```

        oTile = CreateObject(OBJECT_TYPE_PLACEABLE,
"plc_invisobj", lLoc, FALSE, "x2_tmp_tile");
        ApplyEffectToObject(DURATION_TYPE_PERMANENT,
EffectVisualEffect(nGroundTile), oTile);
        SetPlotFlag(oTile,TRUE);
        // next row
        vPos.y = vPos.y + 10.0;
    }
    // next column, first row
    vPos.x = vPos.x + 10.0;
    vPos.y = fY;
}
}
void TLResetAreaGroundTileRange(object oArea, int nColumn, int
nRow, int nColumns=1, int nRows=1)
{
    object oTile;
    vector vPos;
    // sanity check
    if(nColumns < 1 || nRows < 1) return;
    // get bounds
    float fX1 = 10.0 * nColumn;
    float fY1 = 10.0 * nRow;
    float fX2 = 10.0 * (nColumn + nColumns);
    float fY2 = 10.0 * (nRow + nRows);
    // reset range
    oTile = GetFirstObjectInArea(oArea);
    while (GetIsObjectValid(oTile))
    {
        // check object is a tile
        if (GetObjectType(oTile) == OBJECT_TYPE_PLACEABLE &&
GetTag(oTile) == "x2_tmp_tile")
        {
            // check tile is located in range
            vPos = GetPositionFromLocation(GetLocation(oTile));
            if(vPos.x > fX1 && vPos.x < fX2 && vPos.y > fY1 &&
vPos.y < fY2)
            {
                SetPlotFlag(oTile,FALSE);
                DestroyObject (oTile);
            }
        }
        oTile = GetNextObjectInArea(oArea);
    }
}
}

```

On Hit Cast Spell

By Georg Zoeller

Note: If you want to add an item with a custom OnHitCastSpell property, I really suggest you take the template provided in **x2_it_example.nss**

1) How does it work?

When an OnHitCastSpell is fired, the original spells spell script gets fired, the variables are distributed this way (as explained in x2_s3_onhitcast.nss):

OBJECT_SELF

The possessor of the item firing the spell

GetSpellTargetObject()

Weapon: The person hit

Armor/Shield: The person hitting the armor/shield

GetCasterLevel()

The caster level set on the item property

GetSpellCastItem()

The item triggering the ability

2) Why are there no beneficial spells (i.e. heal) in the list?

See 1. It would not make sense to heal the person you are hitting, but you can script them using OnHit Unique power.

3) Is it possible to do a percentage chance for the OnHit spell?

Yes, it just requires a bit custom coding. In fact there are a couple of OnHitCastSpell item properties that have such a percentage chance coded in:

Check the Chaos Shield on the palette - It's percentage chance to trigger its OnHit property is equal to 2.5 times the caster level you set the item property to.

This means if you put the OnHitCastSpell: Chaos Shield property CasterLevel 20 on an armor or shield, it would have a $2.5 \times 20 = 50\%$ chance of triggering it's spellsript on each hit

4) "But people will abuse this in modules, it will unbalance the game and make <put class here> obsolete?"

To be honest, I could not care less.

Its up to the individual module designer to balance his/her module or persistent world. IF the designer thinks giving a monty haul of OnHitCastSpell properties out, its his decision, and he will pay for it as people will just move to another world to play on. As for LocalVault ... there is a reason why we also have ServerVault mode in the game.

5) "Why is this property so dangerous performance wise - that sounds like bad programming"

Just think of this: Each time a creature wearing an armor or shield with this property is being hit, and each time a creature wielding a weapon with this property hits, a spellscript is fired. These spellscripts can be rather complex and thus they take some time to execute. Usually this is a no brainer, as it does not happen every few seconds that these spells go off, but once you put them as OnHitCastSpell property, they will execute very often.

Magic missile has a 28 ms execution time on my test computer. Envision a fighter doing 5 attacks per round against a weak opponent. This would make 28x5 ms cpu time required in addition to all other attack engine computations and scripts (OnAttack). Now just think about haste or whirlwind. If you make a rather bad choice about your OnHit spells (i.e. an area of effect spell that needs to loop over a huge number of creatures), only a hand full of people with such a weapon fighting each other have the potential to take up most of your CPU cycles:

Improved Whirlwind with a OnHitCastSpell: IGMS script executed in a group of 20 goblins would cause something like 100ms (guessed) x 20 = 2000 ms of script executions each combat round, **in addition** to probably 20 OnDamage and OnHit scripts firing.

Bottom-line: The individual module / world designer needs to act responsible when placing weapons or armor with this property. If given out rarely or using its own, optimized custom scripts (that could even be cached), you will most likely not run into trouble and be able to add a great deal of interesting new stuff to your world.

People running local vault servers should make use of the new item property scripting commands to strip any unwanted OnHitCastSpell item property from incoming character's weapons.

6) "Any way to find out if the OnHitCastSpell event was triggered by a critical?"

no

7) Using the item event routing with OnHitCastSpell

The item event routing system that is included in the new module default scripts that are attached to any new module created with the XP2 toolset includes an event for the OnHitCastSpell event.

See x2_it_example.nss and the sticky in the scripting forum for more details

8) Under which conditions does this event fire

OnHitCastSpell only fires if:

- a) at least one point of damage is caused
- b) the attacker and attackee are both creatures

it does not fire for spells.

Intelligent Weapons

By Georg Zoeller and Misc.

The talking sword was scripted with the community builders in mind, so it's not too difficult.

How to create a new talking weapon:

- 1) copy enserrics dialog file (x2_iw_enserric) and save it as i.e. my_iw_sword.
- 2) remove all of enserrics story related dialog, so only battle cries and hit sounds are left in the dialog tree. make sure you leave at least one line without any conversation condition so people can actually talk to your sword.
- 3) Attach the OnHit: Intelligent weapon (makes the weapon speak on hit) and OnHit:TalkTo (unlimited times per day) properties to the weapon
- 4) change the weapon's tag to the name of your dialog file.

That's it, you now have your own talking weapon

Kylania:

Noticed an odd thing about this. When you TalkTo the sword it'll have the name "Enserric the Sword" since the invisible placeable it spawns for you to talk to is hard coded into the x2_inc_intweapon script.

```
x2_inc_intweapon Line 183:  
    object oCreate = CreateObject(OBJECT_TYPE_PLACEABLE,  
    "x2_plc_intwp", GetLocation (OBJECT_SELF));
```

Maybe you'd want to replace that with say the name of your placed sword and create a matching invis object for it?

Or go with the "I used to be dead" concept and instead substitute a Ghostly Image of the previous life of the sword, like an old man in chain mail or something?

Conclusion by EgoAnt

Step 1: Create the item, it doesn't really matter what it is, just remember that if you want it to be a weapon that talks in battle there are some extra steps. For the purposes of this example I created a robe. As FaxCelestis said above, in order to be able to talk to the item you will need to add the "Talk To" property found under "Cast Spell"

Step 2: In the standard placeable objects list find the "Enserric the sword" placeable object. Edit a copy of it and define a new name, portrait, and blueprint ResRef. Make a

note of the new ResRef.

Step 3: Open the following resources:

x2_inc_intweapon

x2_s3_intitemtlk

In the x2_inc_intweapon go to line 183 and change "x2_plc_intwp" to your new ResRef (as noted above). Save this file.

Now all you have to do is put a single space anywhere in the "x2_s3_intitemtlk" file and then delete the space you just made. This will signal to the toolset that this file needs to be recompiled. Save the file.

Step 4: Follow Georg Zoeller's instructions from above to create the conversation. However, feel free to add in any dialogue you feel is appropriate to the sword. In my case I was making a robe, so for this step instead of copying the existing dialog file I simply created a brand new one and named it "pwr_robeshollow". The reason for this, of course, is that the player isn't going to be running around bashing things with his robe... 😊

For those too lazy to scroll back up (and in the interest of having all the instructions together) here are those instructions again:

- 1) copy enserrics dialog file (x2_iw_enserric) and save it as i.e. my_iw_sword.
- 2) remove all of enserrics story related dialog, so only battle cries and hit sounds are left in the dialog tree. make sure you leave at least one line without any conversation condition so people can actually talk to your sword.
- 3) Attach the OnHit: Intelligent weapon (makes the weapon speak on hit) and OnHit:TalkTo (unlimited times per day) properties to the weapon

Step 5: Start testing, it should pretty much work at this point.

Adding Material Components to the Crafting Feats

By Georg Zoeller

Technical Stuff:

The Crafting Feats System in HotU offers the option for module builders to add material components to any spell to balance out the economical impact of these feats on the module.

There is a 2da file called **des_matcomp.2da**.

It only needs to exist on the serve side, so you can put it into the server version of your

hakpak or even your override directory (which would also affect single player).

Inside that file there is a comp_tag column. In this column you can put the tag of an item that must be present in the player's inventory when he wants to cast the spell. If you make the column empty (****), the spell does not need a material component.

We restricted a couple of spells to make them unavailable for scrolls for balancing reasons (and that's not only true for multiplayer, chain casting timestop will wreck balance in any campaign).

The three spells that require a material component by default are:

Harm

Timestop

True Seeing

The material component for all these spells is NW_IT_MSMLMISC17 (Dragon Blood) which is usually dropped by dragons.

How to Disable Crafting Feats

By Georg Zoeller

Craft Skills : Craft Weapon, Craft Armor

You can disable the craft skill conversation that allows a character to modify armor. You can disable these skills on a per module, per area (i.e. if you don't think people should modify their armor in a dungeon) or per character basis (just set the variable on either PC, Module or Area):

```
int "X2_L_DO_NOT_ALLOW_CRAFTSKILLS" = TRUE
```

Remember you can use the convenient new "Variables" window in the toolset to do this for areas or the module.

Note: This only disables the craft part of the crafting conversation. It does NOT effect the ability to craft weapons or armor from components, if you want to restrict that, restrict access to the components.

Craft Feats: Brew Potions, Scribe Scrolls, Craft Wands

By setting the following module switch (see x2_mod_def_load.nss for details), you can disable the crafting feats in your module:

```
MODULE_SWITCH_DISABLE_ITEM_CREATION_FEATS
```

HOU Problems With Scripts That Use the DelayCommand

By Georg Zoeller

<http://nwn.bioware.com/forums/viewtopic.html?topic=290730&forum=47>

Dynamic item properties

By Georg Zoeller

Example: Subraces

The following example demonstrates the use of dynamic item properties in creating a flexible, completely data (2da) driven subrace system.

I plan on add a "add a custom made feat to your new subraces" kind of tutorial in the future, but since I am doing this in my free time, it might be a while 🍷

Technical Overview / Features

This system works by creating a creature hide on entering player characters. The creature hide then gets the item properties added for the subrace the player chose when creating his character.

Feature set:

- * completely data driven with all data residing on the server
- * add and remove subraces easily without changing code or recompiling
- * enforces the players subrace field to be valid
- * restrict subraces to member of certain races (i.e. only elves can become drow's)

"_inc_subraces.nss"

This file is the heart of the subrace system. Please note that it is a part of a much larger module I am playing with right now, so I needed to pull some functions from other files into it to make it run independently.

```
//-----  
//                               S u b r a c e   D e m o  
//-----  
/*  
                               _inc_subraces.nss  
The purpose of this file is to demonstrate how to make use of dynamic item  
properties and the bioware item property include file to create a highly  
flexible, 2da driven subrace system.  
Subrace 2DA Definition  
Row      Field      Purpose  
0        id         Unique Identifier (0 = no subrace)  
1        hide       reference to creature hide item (**** = x2_it_emptyskin)
```

```

    2      properties Comma separated list of properties to add to the hide
    3      race      Racial type this subrace is limited to (-1 = no limit)
*/
//-----
// author: georg zoeller; version: 2003-Oct-21; (c) 2003 Bioware Corp.
//-----

// * Include the Bioware item property include.
#include "x2_inc_itemprop"
// #include "_inc_core"
//-----
//                               C O N S T A N T S
//-----
// * Set to TRUE if you feel you have too much memory and want to speed up 2da
reads
const int TD_CFG_PERF_CACHE_ALL_2DA_READS = FALSE;

//-----
//                               C O N S T A N T S
//-----
const int GZ_SUBRACE_2DAROW_ID      = 0;
const int GZ_SUBRACE_2DAROW_HIDE   = 1;
const int GZ_SUBRACE_2DAROW_PROPS  = 2;
const int GZ_SUBRACE_2DAROW_RACELIMIT = 3;

const int GZ_SUBRACE_ERROR_UNKNOWN = -1;
const int GZ_SUBRACE_ERROR_INVALID_RACE = -2;
//-----
//                               C O N F I G U R A T I O N
//-----
// * Use these constants to change the names of the 2da files that hold all
// * information on subraces and properties
const string GZ_SUBRACE_CONST_PROP_2DA = "gz_properties";
const string GZ_SUBRACE_CONST_DEF_2DA = "gz_subraces";
//-----
//                               I N T E R F A C E
//-----
// * Returns the creature hide object used for managing the subrace properties
// * on the player. If the player does not have a creature yet, create and
// * equip one, before returning it.
object SubraceGetOrCreateCreatureHide(object oPC, string sSubRace);
// * Returns the item property defined in row nPropRow of the 2da file in
// * GZ_SUBRACE_CONST_PROP_2DA
itemproperty SubraceGetItemPropertyFrom2DA(int nPropRow);
// * Reads GZ_SUBRACE_CONST_DEF_2DA and returns value for sSubrace in n2DARow
// * n2DARow possible values:
// *   GZ_SUBRACE_2DAROW_ID      - Subrace ID (unique id)
// *   GZ_SUBRACE_2DAROW_HIDE   - ResRef of Hide Item
// *   GZ_SUBRACE_2DAROW_PROPS  - Comma separated properties list
// *   GZ_SUBRACE_2DAROW_RACELIMIT - Racial Type Limitation
string SubraceGetRaceData(string sSubrace, int n2DARow);
// * Perform the subrace check and perform all necessary operations on the
// * player to make the subrace found work on the character. If no valid subrace
// * is found, reset the players subrace field to a null-string
// * This function is intended to be called from the OnClientEnter event of a
// * module but will work from anywhere else as well.
int SubraceDoSubraceCheck(object oPC);

//-----
//                               I M P L E M E N T A T I O N
//-----

//-----
//                               +++ I M P O R T E D   f r o m   _ i n c _ d e b u g   +++
//-----
const int TD_DEBUG = TRUE;
const int TD_DEBUG_WARNINGS = 1; // 1 - all, 2 - log only
void DebugMsg(string sMessage, string sDomain = "[Global]", string sFunction =
"Unspecified");
void DebugWriteLog(string sMessage, string sDomain = "[Global]", string sFunction
= "Unspecified");

```

```

void DebugWriteWarning(string sMessage, string sDomain = "[Global]", string
sFunction = "Unspecified");
void LogWriteMessage(string sMessage, string sDomain = "[Global]", string
sFunction = "Unspecified");
void LogWriteWarning(string sMessage, string sDomain = "[Global]", string
sFunction = "Unspecified");
void DebugWriteLog(string sMessage, string sDomain = "[Global]", string sFunction
= "Unspecified")
{
    if (TD_DEBUG)
    {
        WriteTimestampedLogEntry(sDomain + "::"+sFunction+ "() - " + sMessage);
    }
}
void DebugWriteWarning(string sMessage, string sDomain = "[Global]", string
sFunction = "Unspecified")
{
    if (TD_DEBUG_WARNINGS>0)
    {
        WriteTimestampedLogEntry("WARN: " + sDomain + "::"+sFunction+ "() - " +
sMessage);
    }
}
void DebugMsg(string sMessage, string sDomain = "[Global]", string sFunction =
"Unspecified")
{
    if (TD_DEBUG)
    {
        SendMessageToPC(GetFirstPC(),sDomain + "::"+sFunction+ "() - " + sMessage);
    }
    DebugWriteLog(sMessage, sDomain, sFunction ) ;
}
void DebugWarning(string sMessage, string sDomain = "[Global]", string sFunction =
"Unspecified")
{
    if (TD_DEBUG_WARNINGS == 1)
    {
        SendMessageToPC(GetFirstPC(),"WARN " +sDomain + "::"+sFunction+ "() - " +
sMessage);
    }
    DebugWriteWarning(sMessage, sDomain, sFunction ) ;
}
void LogWriteMessage(string sMessage, string sDomain = "[Global]", string
sFunction = "Unspecified")
{
    WriteTimestampedLogEntry(sDomain + "::"+sFunction+ "() - " + sMessage);
}
void LogWriteWarning(string sMessage, string sDomain = "[Global]", string
sFunction = "Unspecified")
{
    WriteTimestampedLogEntry("WARN: " + sDomain + "::"+sFunction+ "() - " +
sMessage);
}

//-----
//                                     +++ I M P O R T E D   from _inc_tools +++
//-----
string GZGet2DAString(string s2DA, string sCol, int nRow, int bAlwaysCache =
FALSE)
{
    int bCache = (bAlwaysCache || TD_CFG_PERF_CACHE_ALL_2DA_READS);
    string sCache = "GZ_2DA_" +s2DA + "+" +sCol + "+" + IntToString(nRow);
    string sRet = GetLocalString(GetModule(),sCache);
    if (sRet != "")
    {
        DebugWriteLog("HIT: " + sCache+": " + sRet , "_inc_tool","GZGet2DAString");
        return sRet;
    }
    DebugWriteLog("MISS: " + sCache,"_inc_tool","GZGet2DAString");
    sRet = Get2DAString(s2DA,sCol, nRow);
    if (sRet != "" && bCache)

```

```

    {
        SetLocalString(GetModule(),sCache,sRet);
        DebugWriteLog("Write: " + sCache + ": " +
sRet, "_inc_tool", "GZGet2DAString");
    }
    return sRet;
}
int GZGet2DAInt(string s2DA, string sCol, int nRow, int bAlwaysCache = FALSE)
{
    int nRet = StringToInt(GZGet2DAString(s2DA, sCol, nRow, bAlwaysCache));
    return nRet;
}
//-----
//                                     +++ I M P O R T E D   from _inc_player +++
//-----

void PlayerSendStrRef(object oPlayer, int nStrRef)
{
    SendMessageToPCByStrRef(oPlayer, 0x01000000 +nStrRef);
}

//-----
//                                     I M P L E M E N T A T I O N
//-----

//-----
// Georg Zoeller, Oct 2003
// Returns the creature hide object used for managing the subrace properties
// on the player. If the player does not have a creature hide yet, create and
// equip one, before returning it. If the player has a creature hide, it's
// resref is compared against the definition in the 2da and it is replaced if
// the resref's don't match
//-----
object SubraceGetOrCreateCreatureHide(object oPC, string sSubRace)
{
    //-----
    // Get definition of creature hide
    //-----
    string sRefHide = GZGet2DAString( GZ_SUBRACE_CONST_DEF_2DA, sSubRace,
GZ_SUBRACE_2DAROW_HIDE);
    object oHide = GetItemInSlot(INVENTORY_SLOT_CARMOUR,oPC);
    int bOverride;
    if (GetResRef(oHide) != sRefHide)
    {
        DestroyObject(oHide);
        bOverride = TRUE;
    }
    //-----
    // If there is no creature hide already create one...
    //-----
    if (!GetIsObjectValid(oHide) || bOverride)
    {
        //-----
        // Create creature hide, based on 2da definition
        //-----
        if (sRefHide == "")
        {
            //-----
            // Empty String in the 2da, default to empty creature hide
            //-----
            sRefHide = "x2_it_emptyskin";
        }
        oHide = CreateItemOnObject(sRefHide, oPC);
        //-----
        // Some NWN creature hides are inv
        //-----
        SetIdentified(oHide,TRUE);
        AssignCommand(oPC,ActionEquipItem(oHide,INVENTORY_SLOT_CARMOUR));
    }
    return oHide;
}

```

```

}
//-----
// Georg Zoeller, Oct 2003
// Returns the item property defined in row nPropRow of the 2da file in
// GZ_SUBRACE_CONST_PROP_2DA
//
// Note: I wrapped ItemProperty ID #0 to 255, so I can use 0 as invalid
//-----
itemproperty SubraceGetItemPropertyFrom2DA(int nPropRow)
{
    int nPropID = GZGet2DAInt("gz_properties", "PropertyID", nPropRow);
    itemproperty ip;
    int nParam1;
    int nParam2;
    int nParam3;
    int nParam4;
    if (nPropID>0)
    {
        //-----
        // We interprete 255 as 0 so we can use 0 as invalid before this point
        //-----
        if (nPropID == 255)
        {
            nPropID = 0;
        }
        nParam1 = GZGet2DAInt(GZ_SUBRACE_CONST_PROP_2DA, "Param1", nPropRow);
        nParam2 = GZGet2DAInt(GZ_SUBRACE_CONST_PROP_2DA, "Param2", nPropRow);
        nParam3 = GZGet2DAInt(GZ_SUBRACE_CONST_PROP_2DA, "Param3", nPropRow);
        nParam4 = GZGet2DAInt(GZ_SUBRACE_CONST_PROP_2DA, "Param4", nPropRow);
        //-----
        // Generate the item property from the parameters
        //-----
        ip = IPGetItemPropertyByID(nPropID,nParam1, nParam2, nParam3, nParam4);
    }
    if (!GetIsItemPropertyValid(ip))
    {
        //-----
        // Some message into the log to make debugging easier
        //-----
        DebugWriteWarning("_inc_subraces::SubraceGetItemPropertyFrom2DA - invalid
item property created, values: " +
                        "ID: " + IntToString(nPropID) + " p:" +
IntToString(nParam1) + ", " + IntToString(nParam2) + ", " +
                        IntToString(nParam3) + ", " +
IntToString(nParam4), "_inc_subraces", "SubraceGetItemPropertyFrom2DA" );
    }
    return ip;
}

//-----
// This is the heart of the subrace system. This function determines if the pc
// has a valid subrace, meets all requirements for that subrace and, and finally
// adds the subrace specific item properties to the player's creature hide slot
// The parameter sSetSubraceTo can be specified to overwrite the player's
// current value in the SubRace field, effectively forcing another subrace.
//-----
int SubraceApplyPlayerSubrace(object oPC, string sSetSubraceTo = "")
{
    //-----
    // If sSetSubraceTo was set, overwrite existing subrace with a new one
    //-----
    if (sSetSubraceTo != "")
    {
        SetSubRace(oPC,sSetSubraceTo);
    }
    //-----
    // Determine the 2da row and retrieve the subrace id
    //-----
    string sSubRace = GetStringLowerCase(GetSubRace(oPC));
    int nSubraceId = GZGet2DAInt(GZ_SUBRACE_CONST_DEF_2DA, sSubRace,
GZ_SUBRACE_2DAROW_ID,TRUE );
}

```

```

//-----
// Get the PCs creature hide object.
//-----

if (nSubraceId == 0 && sSubRace != "")
{
    return GZ_SUBRACE_ERROR_UNKNOWN ;
}
if (nSubraceId>0)
{
    //-----
    // Enforce Racial Restrictions if existant.
    //-----
    int nReqRacialType = GZGet2DAInt( GZ_SUBRACE_CONST_DEF_2DA, sSubRace,
GZ_SUBRACE_2DAROW_RACELIMIT );
    if (nReqRacialType != -1)
    {
        if (GetRacialType(oPC) != nReqRacialType)
        {
            return GZ_SUBRACE_ERROR_INVALID_RACE;
        }
    }
    object oHide = SubraceGetOrCreateCreatureHide(oPC,sSubRace);
    itemproperty ip;
    //-----
    // Determine and add all item properties that define this subrace
    // to the creature hide
    //-----
    string sProps = GZGet2DAString( GZ_SUBRACE_CONST_DEF_2DA, sSubRace,
GZ_SUBRACE_2DAROW_PROPS);
    string sTemp;
    int nTemp;
    int nPos = FindSubString(sProps,",");
    while (nPos!= -1)
    {
        //-----
        // if multiple properties have been specified seperated by ,
        // add all of them
        //-----
        sTemp = GetSubString(sProps,0,nPos);
        nTemp = StringToInt(sTemp);
        sProps = GetSubString(sProps,nPos+1, GetStringLength(sProps) -
GetStringLength(sTemp)-1);
        nPos = FindSubString(sProps,",");
        if (nTemp >0)
        {
            //-----
            // Retrieve item property and add it to the hide, ignoring any
            // existing item property of the same type
            //-----
            ip = SubraceGetItemPropertyFrom2DA(nTemp);
            IPSafeAddItemProperty(oHide, ip, 0.0f,
X2_IP_ADDPROP_POLICY_IGNORE_EXISTING, FALSE,FALSE);
        }
        else
        {
            nPos = -1;
        }
    }
    nTemp = StringToInt(sProps);
    if (nTemp >0)
    {
        //-----
        // Retrieve item property and add it to the hide, ignoring any
        // existing item property of the same type
        //-----
        ip = SubraceGetItemPropertyFrom2DA(nTemp);
        IPSafeAddItemProperty(oHide, ip, 0.0f,
X2_IP_ADDPROP_POLICY_IGNORE_EXISTING, FALSE,FALSE);
    }
}
}

```



```

//-----
// return the subrace id to the calling script. 0 indicates that no valid
// subrace was found
//-----
return nSubraceId;
}
//-----
// Reads GZ_SUBRACE_CONST_DEF_2DA and returns value for sSubrace in n2DARow
// n2DARow possible values:
// GZ_SUBRACE_2DAROW_ID - Subrace ID (unique id)
// GZ_SUBRACE_2DAROW_HIDE - ResRef of Hide Item
// GZ_SUBRACE_2DAROW_PROPS - Comma seperated properties list
// GZ_SUBRACE_2DAROW_RACELIMIT - Racial Type Limitation
//-----
string SubraceGetRaceData(string sSubrace, int n2DARow)
{
    return GZGet2DAString( GZ_SUBRACE_CONST_DEF_2DA, sSubrace, n2DARow);
}
//-----
// Perform the subrace check and perform all necessary operations on the
// player to make the subrace found work on the character. If no valid subrace
// is found, reset the players subrace field to a null-string
// This function is intended to be called from the OnClientEnter event of a
// module but will work from anywhere else as well.
//-----
int SubraceDoSubraceCheck(object oPC)
{
    int nRet = SubraceApplyPlayerSubrace(oPC);
    if (nRet == GZ_SUBRACE_ERROR_UNKNOWN )
    {
        SendMessageToPC(oPC,"Unknown subrace selected");
        SetSubRace(oPC,"");
    }
    else if (nRet == GZ_SUBRACE_ERROR_INVALID_RACE)
    {
        SendMessageToPC(oPC,"Your subrace can not be member of the race you
selected");
        SetSubRace(oPC,"");
    }
    else if (nRet != 0)
    {
        SendMessageToPC(oPC,"Valid subrace detected - character adjusted.");
    }
    return nRet;
}

```

Module OnEnter Script

Put the following code in the module's OnClientEnter Script to make the subrace system take effect:

```

// This initializes subrace support on the player.
// Detailed documentation provided within _inc_subraces.nss
//-----
SubraceDoSubraceCheck(oPC);

```

gz_subraces.2da

This file is holds the definition for each subrace. It is a design side only 2da file, thus it needs to be only present on the server. Please note that the data in this file is for demonstration purposes only, it does not contain the appropriate data for each of the subraces

```

2DA V2.0
Label          drow          tiefling
0 id           1             2
1 hide        ****          nw_it_creitem196

```

```

2  properties  7,27      ****
3  requireRace 1        6

```

Explanation:

For each subrace you want to have, you need a new column in this 2da, that uses the lowercase name of the subrace as name. The rows in this 2da file are defined as follows:

0 - id - a unique ID for your subrace, which MUST NOT be 0

1 - hide - the resref to the creature hide item. if you leave this field empty(****), the game will automatically create an empty skin

2 - properties - a comma seperated list of item properties that are going to be added to the creature hide. No spaces are allowed between the numbers and commas. Each number represents a line in gz_properties.2da

3 - requireRace - A race id (row number from races.2da) that is allowed to take this subrace. I.e. by putting a 1 [elf] into this row, only elves can take this subrace. If anyone else enters this subrace into their subrace field, it is automatically reset.

gz_properties.2da

This file is used to define the item properties that the system knows. The line indices for those numbers are put into the properties row of gz_subraces.2da

```

2DA V2.0
Label          PropertyID  Param1  Param2  Param3  Param4  Param5
0  ****          ****          ****    ****    ****    ****    ****
1  FtWhirlwind   12          29      ****    ****    ****    ****
2  FtDisarm      12          28      ****    ****    ****    ****
3  AbilityStr+1  255         0        1      ****    ****    ****
4  AbilityStr+2  255         0        2      ****    ****    ****
5  AbilityStr+3  255         0        3      ****    ****    ****
6  AbilityStr+4  255         0        4      ****    ****    ****
7  AbilityDex+1  255         1        1      ****    ****    ****
8  AbilityDex+2  255         1        2      ****    ****    ****
9  AbilityDex+3  255         1        3      ****    ****    ****
10 AbilityDex+4  255         1        4      ****    ****    ****
11 AbilityCon+1  255         2        1      ****    ****    ****
12 AbilityCon+2  255         2        2      ****    ****    ****
13 AbilityCon+3  255         2        3      ****    ****    ****
14 AbilityCon+4  255         2        4      ****    ****    ****
15 AbilityInt+1  255         3        1      ****    ****    ****
16 AbilityInt+2  255         3        2      ****    ****    ****
17 AbilityInt+3  255         3        3      ****    ****    ****
18 AbilityInt+4  255         3        4      ****    ****    ****
19 AbilityWis+1  255         4        1      ****    ****    ****
20 AbilityWis+2  255         4        2      ****    ****    ****
21 AbilityWis+3  255         4        3      ****    ****    ****
22 AbilityWis+4  255         4        4      ****    ****    ****
23 AbilityCha+1  255         5        1      ****    ****    ****
24 AbilityCha+2  255         5        2      ****    ****    ****
25 AbilityCha+3  255         5        3      ****    ****    ****
26 AbilityCha+4  255         5        4      ****    ****    ****
27 AbilityStr-1  27          0        1      ****    ****    ****
28 AbilityStr-2  27          0        2      ****    ****    ****
29 AbilityDex-1  27          1        1      ****    ****    ****
30 AbilityDex-2  27          1        2      ****    ****    ****
31 AbilityCon-1  27          2        1      ****    ****    ****
32 AbilityCon-2  27          2        2      ****    ****    ****
33 AbilityInt-1  27          3        1      ****    ****    ****
34 AbilityInt-2  27          3        2      ****    ****    ****

```

35	AbilityWis-1	27	4	1	****	****	****
36	AbilityWis-2	27	4	2	****	****	****
37	AbilityCha-1	27	5	1	****	****	****
38	AbilityCha-2	27	5	2	****	****	****
39	DamRes_Fire5	23	10	1	****	****	****
40	DamRes_Fire10	23	10	2	****	****	****
41	DamRes_Fire15	23	10	3	****	****	****
42	DamRes_Elec5	23	9	1	****	****	****
43	DamRes_Elec10	23	9	2	****	****	****
44	DamRes_Elec15	23	9	3	****	****	****

Explanation

Col:PropertyId - This column holds the id of an item property (ITEM_PROPERTY_* constant from nwscript.nss).

Example: 12 is ITEM_PROPERTY_BONUSFEAT

Col:Param1-5 - These columns hold all the parameters that would go into the appropriate item property function that would be called to create the item property defined in the PropertyId column.

Example:

If your ItemPropertyId is 12 (ITEM_PROPERTY_BONUSFEAT), you only need to put one parameter (the id from iprp_feats) into these columns, as the scripting function ItemPropertyBonusFeat(int nFeat) only has one parameter (nFeat).

further uses/ideas

- * This system probably could be used to some kind of lycanthropy scripts.
- * You can script conversations that change the players subrace and use the functions provided in the file (SubraceApplyPlayerSubrace) to easily script these conversations.
- * You can add custom feats in HotU - allowing you to make even more convincing subraces (i.e. drows that can cast darkness).

Notes

Please note that if your server is running the Enforce Legal Character option (ELC), hides equipped on a player will be removed when the player enters (?). You would need to call the subrace check on every player login to avoid that.

What this example is not

This example is not a subrace system that circumvents the existing limitations of NWScript, namely ECL compliant subraces or ability score adjustments on the actual character data. If you are looking for a way to break the barriers of NWScript, there are a couple of options for that available [elsewhere](#)

This example is also NOT trying to compete with any of the more complex community implementations of subraces, it is just an example to show how dynamic item properties could be used to remove the need for hardcoding in script.

HotU OnSpawn and Variables

By ddaedelus

How to use HotU OnSpawn & Variables

I hadn't seen anyone post much information on this. Maybe that means it's perfectly clear. I suspect not, however, so I thought I'd put together what I've figured out...

This post hopefully clears up how to use the new OnSpawn and Variables.

What the old (NWN & SoU) OnSpawn required:

To get a creature to spawn in with a certain condition or animation, you had to at least change the OnSpawn script, and depending on what you were doing (for example, a custom OnPerception script) you may even have had to change up to three of the creatures default scripts. This, quite simply, was a huge pain.

What the new (HotU) OnSpawn does:

You've probably noticed that if you right click on a placed creature, one of the choices that comes up is "Variables." The HotU OnSpawn script uses these variables to determine spawn-in conditions; the result being, you don't have to alter the default OnSpawn. In fact, the only script you may need to alter is the UserDefined script, and that's only if you're doing something funky.

NOTE: If you updated your module to HotU in the middle of construction, any **pre-update** creatures you have placed will have the SoU OnSpawn script, which doesn't use Variables (setting variables on these creatures will have no effect). Any **new** creatures you create or place from the standard palette will use the HotU OnSpawn script (which is called "x2_def_spawn"), and can use the new Variables feature.

Using either OnSpawn script will work in HotU, so there's no need to go back and re-work creatures you've already scripted unless you're a consistency freak.

The Variables window:

So to get started on this, place a creature in your module, right-click, and select Variables.

The Window that pops up is pretty basic. The trick to using this window is to know what the variable names are. If you were to open the HotU default OnSpawn script, "x2_def_spawn", you would see that it uses the include file called "x2_inc_switches." The include file is pretty well-commented and mostly self explanatory.

Using the Variables window is most easily demonstrated with a simple example:

Example: A common spawn-in condition is to set the creature to spawn in using its stealth skills, so we're going to use the Variable window to make a creature spawn in hidden.

1. Create the new creature either from scratch or by selecting Edit Copy from a creature in the standard palette.
2. At the bottom of the Advanced tab, you'll see a button called "Variables." Click on that to open the Variables window.
3. Now take a look at the include file. If you scroll down a bit, under the "Creature" heading you'll find that the variable for spawning in hidden is called "X2_L_SPAWN_USE_STEALTH."
4. Back in your Variable window, type "X2_L_SPAWN_USE_STEALTH" (no quotes) into the Name field, and "1" into the Value field. Click on Add. Note that there's a Type field also. This defaults to Int, so in this case we don't need to change it. Check the listing in the "x2_inc_switches" file for the type of variable the switch requires and change it here if necessary.
5. Now save your new creature. And quite honestly, that's all there is to it. Much easier than in SoU.

I didn't make my creature in the palette so need to update my blueprint.

I make it a general habit for all my creatures to be in the palette so that I can use them in encounter triggers. I would recommend you do the same. Also I would strongly suggest that any editing you do to a creature be done in the palette rather than on a placed instance. Why?

If you placed a creature in your area, then modified the variables (or anything else) there, and then want to add it to your custom palette (or update your palette from the instance), you're in for a bit of grief: The toolset will try to make custom blueprints of every item your creature is carrying **EVEN IF YOU DIDN'T MODIFY THEM** in the instance. This is exceptionally annoying.

If you do this, the workaround is to X out of every pop-up blueprint window that comes up (except for the creature itself). There will be one for every item the creature is carrying. If you hit OK instead, it will alter the resref of the item. That's potentially bad if you call that resref from any script in your module.

Real solution: Don't edit placed instances of creatures. Always edit from the palette and then update instances from there.

What about UserDefined scripts that I want to fire from OnSpawn?

Look at the top commented portion of "x2_def_spawn." You'll see this:

```
    If you set a ninteger on the creature named
    "X2_USERDEFINED_ONSPAWN_EVENTS"
    The creature will fire a pre and a post-spawn
    event on itself, depending on the value of that
    variable
    1 - Fire Userdefined Event 1510 (pre spawn)
    2 - Fire Userdefined Event 1511 (post spawn)
    3 - Fire both events
```

Now look at the bottom of the default UserDefined script. You'll see this (I've added the comments to it):

```
    else if (nUser == EVENT_USER_DEFINED_PRESPAWN)
    {
    // This part of the UserDefined script fires if in the variables
    window you set
    // "X2_USERDEFINED_ONSPAWN_EVENTS" to either 1 or 3. Don't be
    confused
    // by the "PRESPAWN" part above. That's a local constant.
    }
    else if (nUser == EVENT_USER_DEFINED_POSTSPAWN)
    {
    // This part of the UserDefined script fires if in the variables
    window you set
    // "X2_USERDEFINED_ONSPAWN_EVENTS" to either 2 or 3. Don't be
    confused
    // by the "POSTSPAWN" part above. That's a local constant.
    }
```

As usual, when modifying the UserDefined script, be careful not to override the original. Save it under a different name.

So, what does this all mean? Like above with the sneaky guy, I'll use an example. Say we want to spawn in a creature sitting down on the floor (for a long time).

1. Change the UserDefined script to this:

```
///:////////////////////
///: Name x2_def_userdef
///: Copyright (c) 2001 Bioware Corp.
///:////////////////////
/*
    Default On User Defined Event script
*/
///:////////////////////
///: Created By: Keith Warner
///: Created On: June 11/03
///:////////////////////
const int EVENT_USER_DEFINED_PRESPAWN = 1510;
const int EVENT_USER_DEFINED_POSTSPAWN = 1511;
void main()
{
    int nUser = GetUserDefinedEventNumber();
    if(nUser == EVENT_HEARTBEAT ) //HEARTBEAT
    {
```

```

    }
    else if(nUser == EVENT_PERCEIVE) // PERCEIVE
    {
    }
    else if(nUser == EVENT_END_COMBAT_ROUND) // END OF COMBAT
    {
    }
    else if(nUser == EVENT_DIALOGUE) // ON DIALOGUE
    {
    }
    else if(nUser == EVENT_ATTACKED) // ATTACKED
    {
    }
    else if(nUser == EVENT_DAMAGED) // DAMAGED
    {
    }
    else if(nUser == 1007) // DEATH - do not use for critical
code, does not fire reliably all the time
    {
    }
    else if(nUser == EVENT_DISTURBED) // DISTURBED
    {
    }
    else if (nUser == EVENT_USER_DEFINED_PRESPAWN)
    {
    ActionPlayAnimation(ANIMATION_LOOPING_SIT_CROSS, 1.0,
99999.9);
    }
    else if (nUser == EVENT_USER_DEFINED_POSTSPAWN)
    {
    ActionPlayAnimation(ANIMATION_LOOPING_SIT_CROSS, 1.0,
99999.9);
    }
}
}

```

2. Save it as a different name.

3. Now in the Variables window for the creature enter
“X2_USERDEFINED_ONSPAWN_EVENTS” in the Name field, 3 in the Value field,
and click “Add.”

4. Save the creature. Again, that’s all there is to it.

What’s the difference between a PRESPAWN and POSTSPAWN event?

Good question, and to tell the truth, I have no idea. The whole idea of a PRESPAWN event baffles me. (How are you supposed to trigger an event from something that doesn’t exist in the game yet?)

Notice up in the example UserDefined script where we made the NPC sit that I used the ActionPlayAnimation function in both places and set the variable to 3 so that both the PRE and POST spawn fire. That’s because I couldn’t figure out what the difference was and when to use one and not the other—so I just used both. Safe in this case, though I


```

//:////////////////////
//: Created By: Keith Warner
//: Created On: June 11/03
//:////////////////////
const int EVENT_USER_DEFINED_PRESPAWN = 1510;
const int EVENT_USER_DEFINED_POSTSPAWN = 1511;
void main()
{
    int nUser = GetUserDefinedEventNumber();
    if(nUser == EVENT_HEARTBEAT ) //HEARTBEAT
    {
    }
    else if(nUser == EVENT_PERCEIVE) // PERCEIVE
    {
    }
    else if(nUser == EVENT_END_COMBAT_ROUND) // END OF COMBAT
    {
    }
    else if(nUser == EVENT_DIALOGUE) // ON DIALOGUE
    {
    }
    else if(nUser == EVENT_ATTACKED) // ATTACKED
    {
    }
    else if(nUser == EVENT_DAMAGED) // DAMAGED
    {
    }
    else if(nUser == 1007) // DEATH - do not use for critical code, does not fire
reliably all the time
    {
    }
    else if(nUser == EVENT_DISTURBED) // DISTURBED
    {
    }
    else if (nUser == EVENT_USER_DEFINED_PRESPAWN)
    {
    object oPC = GetFirstPC();
    object oItem1 = GetItemInSlot(INVENTORY_SLOT_RIGHTHAND, oPC);
    object oItem2 = GetItemInSlot(INVENTORY_SLOT_LEFTHAND, oPC);
    // Check to see if the PC is carrying any weapons
    if ((oItem1 == OBJECT_INVALID) && (oItem2 == OBJECT_INVALID))
        // If the PC isn't carrying any weapons, spawn in normally.
        return;
    else
    {
        // If the PC is carrying weapons, change the faction to hostile.
        ChangeToStandardFaction(OBJECT_SELF, STANDARD_FACTION_HOSTILE);
        // Spawn in stealthed. Notice that this is the same variable we
        // would have used in the variables window to spawn a stealthed
        // creature. We're setting it here instead of the variables window
        // because we want to test first to see if it should be set or not. This
        // is where the PRESPAWN event comes in useful.
        // Notice also, that the variables are set on OBJECT_SELF. This fact
        // isn't obvious when using the Variables window.
        SetLocalInt(OBJECT_SELF, "X2_L_SPAWN_USE_STEALTH", 1);
    }
    }
    else if (nUser == EVENT_USER_DEFINED_POSTSPAWN)
    {
    // Were not using a custom POSTSPAWN event.
    }
}
}

```

4. Save this script with a new name.

5. Under the Advanced tab. Click on Variables. Enter X2_USERDEFINED_ONSPAWN_EVENTS in the Name field and 1 in the Value field.

Click Add.

6. Now create a custom encounter using our new stealthy creature and draw it in your area. Place the spawn point and do all the sorts of things you would usually do with an encounter trigger.

7. Try it out.

A few comments on the script above: You may notice while testing it that the spawned in creatures (if you've spawned in more than one) will also attack each other as well as the PC. This is because this isn't really a good use of the `ChangeToStandardFaction` function. Our guard/assassin creatures should have had a custom faction tweaked so that they ignored HOSTILE creatures, and used the `SetReputation` function instead. (In my example we started them out as Merchants who will attack Hostiles.) I only did it this way to make a quick and simple example without having to go through the faction editor.

Georg Zoeller Makes some Notes:

Two things:

1) 1.62 will have an update to `CreateObject` to copy variables over from the templates. This means that variables set in the toolset will now be present on any creature that is created dynamically using `CreateObject`. In 1.61 toolset variables are only present in placed creatures. Yay!

2) You can modify all spawnconditions of a creature by putting an integer variable called "NW_GENERIC_MASTER" on the creature and put the correct value calculated from the values below:

```
int NW_FLAG_SPECIAL_CONVERSATION           = 0x00000001;
int NW_FLAG_SHOUT_ATTACK_MY_TARGET         = 0x00000002;
int NW_FLAG_STEALTH                        = 0x00000004;
int NW_FLAG_SEARCH                         = 0x00000008;
int NW_FLAG_SET_WARNINGS                   = 0x00000010;
int NW_FLAG_ESCAPE_RETURN                  = 0x00000020; //Failed
int NW_FLAG_ESCAPE_LEAVE                   = 0x00000040;
int NW_FLAG_TELEPORT_RETURN                = 0x00000080; //Failed
int NW_FLAG_TELEPORT_LEAVE                 = 0x00000100;
int NW_FLAG_PERCIEVE_EVENT                 = 0x00000200;
int NW_FLAG_ATTACK_EVENT                   = 0x00000400;
int NW_FLAG_DAMAGED_EVENT                  = 0x00000800;
int NW_FLAG_SPELL_CAST_AT_EVENT            = 0x00001000;
int NW_FLAG_DISTURBED_EVENT                = 0x00002000;
int NW_FLAG_END_COMBAT_ROUND_EVENT         = 0x00004000;
int NW_FLAG_ON_DIALOGUE_EVENT              = 0x00008000;
int NW_FLAG_RESTED_EVENT                   = 0x00010000;
int NW_FLAG_DEATH_EVENT                    = 0x00020000;
int NW_FLAG_SPECIAL_COMBAT_CONVERSATION    = 0x00040000;
int NW_FLAG_AMBIENT_ANIMATIONS             = 0x00080000;
int NW_FLAG_HEARTBEAT_EVENT                = 0x00100000;
int NW_FLAG_IMMOBILE_AMBIENT_ANIMATIONS   = 0x00200000;
int NW_FLAG_DAY_NIGHT_POSTING              = 0x00400000;
```

```
int NW_FLAG_AMBIENT_ANIMATIONS_AVIAN      = 0x00800000;
int NW_FLAG_APPEAR_SPAWN_IN_ANIMATION     = 0x01000000;
int NW_FLAG_SLEEPING_AT_NIGHT             = 0x02000000;
int NW_FLAG_FAST_BUFF_ENEMY               = 0x04000000;
```

So if you want a creature to use be in search and stealth mode on spawning and to use ambient animations, you can set "NW_GENERIC_MASTER" to

```
0x00000004 +
0x00000008 +
0x00080000 =
```

```
-----
```

```
0x0008000C = 524300
```

You need to put the value as decimal (the bold value) on the creature. You can use windows calculator (switch to scientific mode) to calculate the hex numbers together and then convert them to decimal mode.

<http://www.statman.info/conversions/hexadecimal.html>

Antimagic Tutorial

By Huntsman29

<http://nwn.bioware.com/forums/viewtopic.html?topic=317782&forum=47>

HotU Wandering Monster System

Checklist by Georg Zoeller

Checklist:

- a) check the default module OnLoad script (x2_mod_def_load)
- b) check the default module OnRest script (x2_mod_def_rest) as well as the rest script used by the official campaign for HotU (I think it's x2_onrest).
- c) IIRC, the rest script actually has an explanation which variables you want to set on an area
- d) You need to check des_restsys.2da (or so), the 2da file which has the "template"

definitions for wandering monsters

e) x2_inc_restsys.nss IS the official rest system. Its commented so reading through it should give you some ideas.

Primarily by RainMan

Finally it's working and since there're a few who still don't understand how this works here comes a (hopefully) easy guide:

Copy and save the following as "des_restsystem.2da" in your override folder:
(see: <http://nwn.bioware.com/forums/viewtopic.html?topic=329600&forum=47&sp=15>)

This is the original HoTU file with a single line added by me (#22). Just modify my line and add new ones and your HoTU single player should keep working pretty fine. Maybe in some future patches this file will be modified by Bioware so be aware that this could break your single player rest system.

Go to your Module properties, Events Tab and edit the "x2_mod_def_load" script which should be located in the "onLoad" event. The script could already be modified and renamed by whoever made the module, in that case i can't help you. You can try to edit the default script and copy the needed part of code to your script. You need to un-comment this line which is one of the last lines in that script:

```
SetModuleSwitch (MODULE_SWITCH_USE_XP2_RESTSYSTEM, TRUE);
```

The next IF instruction will then load the 2da and so initialize the rest system.

Next step is to make sure you are using the "x2_mod_def_rest" script as your onRest event script, well only a part of it is needed. As with the onLoad event you maybe have renamed the script or totally erased, but if not there is not much more to do.

The last thing you need to do is to go in the Advanced tab of every area's property window and add a "string" variable (the little button at the end of the page) called "X2_WM_ENCOUNTERTABLE", set that variable to the name of the encounter you wish to spawn in that area. You can get the name from the "TableName" in the 2da file, some examples: Cave, Undermountain1Central (or Misc which is the line i added). This done you are ready to go. There are a few other variables you can set in the same way. A "int" variable called "X2_WM_AREA_USEDDOORS" which can set the use of doors by the monster spawn routine or another "int" called "X2_WM_AREA_LISTENCHECK" which sets the listen check DC which you need to beat to wake up earlier and not get ambushed.

You don't need the custom 2da for it to work that's optional without it it will default to the stock one which is fairly basic.

- 1) Uncomment line 108 of x2_mod_def_load
- 2) Make sure you have x2_mod_def_load in the OnModuleLoad Event
- 3) Make sure you have x2_mod_def_rest in the OnPlayerRest Event
- 4) Open an area Edit>Properties>Advanced>Variables add this variable X2_WM_ENCOUNTERTABLE string Forest and you will get deer or dogs.

Also you can add WMSetAreaProbability(oArea, 99, 99); at line 35 of x2_mod_def_rest, 99 being the % chance of an encounter(99% for testing purposes).

I added a few lines to my code which enable to set the probability of an encounter in-game, just in case you need the same encounter to fire in one area with it's default probability and in another area with a different probability. It's all based on the "WMSetAreaProbability" function included in the WM system but not used in the onrest event.

Just include this in your onrest script, right before the WM system decides to fire the encounter, should be an IF with this condition: !WMStartPlayerRest(oPC)

```
int iDayProb = GetLocalInt(oArea, "X2_WM_PROBABILITY_DAY");
int iNightProb = GetLocalInt(oArea, "X2_WM_PROBABILITY_NIGHT");
if (iDayProb > -1 && iNightProb > -1)
    WMSetAreaProbability(oArea, iDayProb, iNightProb);
```

Now you can use the two int variables "X2_WM_PROBABILITY_DAY" and "X2_WM_PROBABILITY_NIGHT" to set a value between 0 and 100 in your area properties for a custom spawn probability just in that area.

Be warned that both variables have to be set or the WMSetAreaProbability function won't be called, i didn't need a more powerful code so i didn't bother to code it better than this, though it could easily be "upgraded" to allow only one variable to be set.

Ambient Simulation System

By Ayath

<http://nwwvault.ign.com/View.php?view=Scripts.Detail&id=2433>

VI. Appendix I

How do I close a door automatically after it has been opened?

by Jassper

Closing a door automatically is quite simple. However there is one thing to keep in mind. Closing doors can become very annoying to the players. I suggest only closing doors that are Area Transitions or lead into a large area. Also you will want to make the delay before it closes reasonable. This way PCs aren't constantly opening the same door so they can fight a creature that is on the other side.

Add this script to the doors OnOpen script handle. Change the number 15.0f to whatever. This is the delay in seconds before the door closes. OBJECT_SELF represents the object that the script is attached to. In this case, the door.

```
////////////////////////////////////  
// Auto-Close Door  
////////////////////////////////////  
void main()  
{  
    DelayCommand(15.0f,ActionCloseDoor(OBJECT_SELF));  
}  
// End
```

How do I make my doors Close and lock?

This can be done with the script above with just one added line of code. Keep in mind that the door can't lock until after it is closed. So the Lock command must be delayed as well.

Add this script to the doors OnOpen script handle. Change the number 15.0f to whatever. This is the delay in seconds before the door closes. The Word TRUE will lock the door. FALSE will Unlock the door. OBJECT_SELF represents the object that the script is attached to. In this case, the door.

```
////////////////////////////////////  
// Auto-Close Door / Lock Door  
////////////////////////////////////  
void main()  
{  
    DelayCommand(15.0f,ActionCloseDoor(OBJECT_SELF));  
    DelayCommand(15.5f,SetLocked(OBJECT_SELF,TRUE));  
}  
// End
```

How do I make my NPC Open / Close a Door?

If this is done through a Conversation then is just a matter of applying the script in the “Actions Taken” script handle. NOTE: Remove the nw_walk_wp script from the End Conversation Normally script handle in the convo node. More on this latter.

Place this script in the Actions Taken handle on the text node that you want the NPC to open / close a door. Replace the Word "DOOR_TAG" with the actual TAG of the door you want him to open, include the quotes "". The SetLocked function is included in case the door is locked. Your NPC will walk to the door and Open it. The script to close the Door is very similar, however we must reverse the order of the SetLocked and Action.

```
////////////////////////////////////  
// NPC Open Door  
////////////////////////////////////  
void main()  
{  
    object oDoor = GetObjectByTag("DOOR_TAG");  
    SetLocked(oDoor, FALSE);  
    ActionOpenDoor(oDoor);  
}  
  
////////////////////////////////////  
// NPC Close Door  
////////////////////////////////////  
void main()  
{  
    object oDoor = GetObjectByTag("DOOR_TAG");  
    ActionCloseDoor(oDoor);  
    SetLocked(oDoor, TRUE);  
}
```

How do I make a portal?

A) Portals are nothing more than a scripted transition. To make it work you need a Portal object and a Waypoint. Make the object “usable” and place the waypoint in the location you want your PCs to port to. Give the Waypoint a Unique tag. For this example I’ll call it WP_ARRIVE.

Place the following script in the OnUsed script handle of the portal.

```
////////////////////////////////////  
// Port PC  
////////////////////////////////////  
void main()  
{  
    object oPC = GetLastUsedBy(); // Get the user of the object  
    object oDest = GetObjectByTag("WP_ARRIVE"); // way point  
tag.  
    if(GetIsPC(oPC))  
    {  
        AssignCommand(oPC, JumpToObject(oDest));  
    }  
}
```



```
    }  
}
```

Note: If the Waypoint is in the same area, Henchmen, Familiars, and pets will have a hard time finding their master. That would require additional scripting. If the Waypoint is in a different area, then there should be no problem with followers. They should make the jump by default.

How do I make my NPC sit.

This is possible many ways. The most common however is to place this line of code in the NPCs OnSpawn script. This assumes your not changing the Tag of the chair. Also make sure the chairs are Non-Static. If you want the change the tag of the default chair then simply replace the word "Chair" in the script with your custom tag. Include the "".

Add this line to the OnSpawn script of the NPC you want to sit. Be sure to re-save the script under a different name.

```
ActionSit( GetNearestObjectByTag( "Chair"));
```

NOTE: Be sure to place this after the WalkWaypoints(); function call. Otherwise the WalkWaypoints may clear his action to sit.

How do I make my NPCs stay seated while talking

This again can be handled several ways. But the simplest is to use a version of the above function in the Root text (first line) of the NPC.

Place this script in the "Actions Taken" script handle on the first line of the NPCs conversation. Note: you must remove the nw_walk_wp script from the "End Conversation Normally" script handle found in the "Current File" Tab or the NPC will stand up at the end of the conversation.

```
////////////////////////////////////  
//Place in "Actions Taken" of the NPCs first convo node  
//Make sure tag is correct and remove the nw_walk_wp  
//script from the End Conversation Normally handle.  
////////////////////////////////////  
void main()  
{  
    object oChair = GetNearestObjectByTag("Chair");  
    //Make sure no one is in chair, if not sit-down.  
    if(!GetIsObjectValid(GetSittingCreature(oChair)))  
    {  
        ClearAllActions(); //This is so he don't spin in his chair,  
following you.  
        ActionSit(oChair);  
    }  
}
```

```
}  
//End
```

How do I take gold from a PC in a conversation? How do I make sure he has enough to Pay?

This is fairly simple to do, the harder part seems to be getting the Conversation right. One thing to keep in mind when doing a conversation is that the line you don't want displayed unless a certain condition is met comes before, lets call it the Default line. Take a look at the example.

```
+Root  
NPC – Hello, it will be 50 gold for a slap in the face.  
PC – Ok. Pay 50 gold.  
NPC – You don't have enough money.  
PC – [End Dialog]  
NPC – Great! <Slap>  
PC – Thanks! [End Dialog]  
PC – No Way![End Dialog]
```

We don't want the line NPC – You don't have enough money. to appear unless he has enough gold to pay. So that's were we will put the check, in the "Starting Condition" of the text node.

Place this script in the "Text Appears When" hook of the text line you don't want to show unless the PC don't have enough money to pay.

```
int StartingConditional()  
{  
    object oPC = GetPCSpeaker();  
    if(GetGold(oPC) < 50) // Set the gold amount you want.  
    {  
        return TRUE; // Return TRUE if he has less than expected  
amount  
    }  
    return FALSE; //return FALSE otherwise. (do not display  
text)  
}
```

If the above script is TRUE then that text node will show to the PC instead of the one below it. If it is FALSE, then the next text line will appear.

Now we need to take the gold, if he has enough. In the "Actions Taken" script handle of the NPC – Greate! <Slap> text node place the following script.

```
////////////////////////////////////  
// Take Gold from PC  
////////////////////////////////////  
void main()
```

```
{
    TakeGoldFromCreature(50, GetPCSpeaker(), TRUE);
}
//End
```

This is a simple script from the script wizard. It takes 50 gold from the PC speaking to the NPC and then destroys it. If you want the NPC to keep the gold then change the TRUE to FALSE.

How do I make my NPC recognize me when I talk to him a second time.

This can all be done in the script wizard. Make your conversation as follows.

```
+Root
NPC – Hello, we meet again.
NPC – Hello, I am NPC and you never met me.
----- PC Ok.
```

On the first text node (Hello, we meet again.) click on the "Text Appears When" tab and click the wizard hat.

Click on the "Local Variables" box and put a mark in it, click next.

In the first blank box, Enter a name like "WeMet" (no quotes). In the next blank box enter a number, 1 is usually good. Click the "Add" button and then "Next" and then done.

Now, in the next Text node, the one you want to appear first (Hello, I am NPC and you never met me.). Click on the "Actions Taken" tab. Click on the little wizard hat and select "Set Local Variables", click Next.

In the first empty box type the SAME NAME as you used in the other wizard. For this example I used "WeMet" (again, no quotes). In the Next blank box enter the SAME NUMBER you gave it the first time, in this case 1. Click "Add" and then Next, then Finish.

Now when your PC talks to the NPC they will get the text node "Hello, I am NPC and you never met me." Your conversation can branch from there. The next time the PC talks to the NPC he will say the first text node, "Hello, we meet again".

NOTE: It is important that you include a PC response. Even if it is a "End Dialog". Scripts that are attached to a NPC's "Actions Taken" script handle *will not* trigger unless a PC has a text line to click on. If there is no PC response line the NPC will say his last line and the Dialog ends, skipping the Actions Taken script.

Why is CreateObject giving me badgers?

One of the finer points about spawning objects using CreateObject is that the "sTemplate" parameter refers to the object's blueprint resref, not the object's tag. Most of BioWare's objects have matching tags and resrefs, but objects and creatures you create yourself in the toolkit probably won't.

What is the blueprint resref, you ask? It's like the tag, but it is unique for every object in

the toolkit's template. It can be found on the "Advanced" tab of your object's properties.

WARNING: If you've created an object using "Edit Copy..." (for example, to make some changes to a previously-existing creature), don't trust what you see on the "Advanced" tab. To get the resref, place your object in the toolset, right-click on it, select edit, and look on the Advanced tab for the resref.

How do I let PCs sit in chairs?

Make the chairs "usable" and place the following script in the "OnUsed" Script handle.

Note: This script can also be placed in the OnUsed of an Invisible Object. This object can then be placed on other things. Place 2 on a bench so more than one person can sit down.

```
////////////////////////////////////  
//Function: Use on placeables so PCs can sit.  
//Ver: 1.0  
//Modified by:  
//Date:  
////////////////////////////////////  
//Place this script in the OnUsed handler of the object  
////////////////////////////////////  
void main()  
{  
    object oChair = OBJECT_SELF;  
    //Make sure no one is in chair, if not sitdown.  
    if(!GetIsObjectValid(GetSittingCreature(oChair)))  
    {  
        AssignCommand(GetLastUsedBy(),ActionSit(oChair));  
    }  
}  
//End
```

NOTE: If the invisible Object is used, Most TileSet Benches, bar stools Don't have a "Front". You will need to play with the position of the invis object to get the PC to sit in the correct direction. If the object is directly on top of (for Example) the Bar stool. PCs will always Face East when they sit on it. Simply slide the invis object off the stool until it falls to the ground. (kinda "half-on / half-off" the stool), The PC should then sit correctly, although they will be sitting on the edge of the stool.

How do I make a Combat Dummy?

In the Toolset, Click on the Monster Icon, then Custom. Right Click on anything, then select "New". Or, In the tool bar Click "Wizards" and select "Creature Wizard". The Creature Wizard should now be open.

Click "Next". Select Human from the List, click next, and then Next again. Click "Select Portrait" and in the Portrait window, click on the "Placeables Objects and Doors". Scroll through the pictures and select either the Archery Target (PLC_FO1_) or the Combat


```
}
```

And place the XP script in the OnDamaged node

```
void main()
{
    int nXP = 2; // amount of xp per damage point
    int nXPMax = 2000; // Max amount of xp a PC can get from the
Dummy
    int nMaxLevel = 3; // Max level the PC can be to still gain
xp from betting up a Dummy
    int nDam = GetTotalDamageDealt();
    object oPC = GetLastDamager();
    if(GetIsPC(oPC) && GetHitDice(oPC) <= nMaxLevel &&
GetLocalInt(OBJECT_SELF,GetName(oPC)) < nXPMax+1)
    {
        GiveXPToCreature(oPC,nDam*nXP);

SetLocalInt(OBJECT_SELF,GetName(oPC),GetLocalInt(OBJECT_SELF,GetN
ame(oPC)) + nDam*nXP);
        // Heal the dummy so it don't get destroyed

ApplyEffectToObject(DURATION_TYPE_INSTANT,EffectHeal(nDam),OBJECT
_SELF);
    }
    else
    {
        SendMessageToPC(oPC,"You have exceeded Level
"+IntToString(nMaxLevel));
        SendMessageToPC(oPC,"Or you have gained
"+IntToString(nXPMax)+" amount of xp");

ApplyEffectToObject(DURATION_TYPE_INSTANT,EffectHeal(nDam),OBJECT
_SELF);
    }
}
```

The script is just a simple/basic XP giving script and it heals the Dummy so it don't get destroyed. Alter / Modify the script as needed.

Note: The PC will be able to move to a different Dummy and start gaining XP again off the new dummy. If you want the PC to only gain nXPMax total. Then use this script.

```
void main()
{
    int nXP = 2; // amount of xp per damage point
    int nXPMax = 2000; // Max amount of xp a PC can get from the
Dummy
    int nMaxLevel = 3; // Max level the PC can be to still gain
xp from betting up a Dummy
    int nDam = GetTotalDamageDealt();
    object oPC = GetLastDamager();
    if(GetIsPC(oPC) && GetHitDice(oPC) <= nMaxLevel &&
GetLocalInt(GetModule(),"Dummy"+GetName(oPC)) < nXPMax+1)
    {
```

```

        GiveXPToCreature(oPC, nDam*nXP);

SetLocalInt(GetModule(), "Dummy"+GetName(oPC), GetLocalInt(GetModule(), "Dummy"+GetName(oPC)) + nDam*nXP);
    // Heal the dummy so it don't get destroyed

ApplyEffectToObject(DURATION_TYPE_INSTANT, EffectHeal(nDam), OBJECT_SELF);
    }
    else
    {
        SendMessageToPC(oPC, "You have exceeded Level
"+IntToString(nMaxLevel));
        SendMessageToPC(oPC, "Or you have gained
"+IntToString(nXPMax)+" amount of xp");

ApplyEffectToObject(DURATION_TYPE_INSTANT, EffectHeal(nDam), OBJECT_SELF);
    }
}

```

Same script. It just recorded the variable differently.

How do I get an NPC to take more than one of the same item from a PC's inventory?

aka "The Goblin-Ear Dilemma"

Good call for this thread -- this one comes up a lot, because the Script Wizard in the conversation editor doesn't really handle cases like this.

What you need to do is loop through every item in the PC's inventory looking for your item, and perform an action on each one you find.

Two functions make it easy to loop through the whole inventory:
GetFirstItemInInventory and **GetNextItemInInventory**.

Here's an example of how to use them to reward a PC for every item they have with the tag "goblin_ear":

```

// This script goes in the "Actions Taken" tab of a
// conversation. It will reward the PC 10 gold and
// 10 XP for every item they possess with the tag
// "goblin_ear".
void main()
{
    // First, get the PC being spoken to
    object oPC = GetPCSpeaker();
    // Now, find the first item in that PC's inventory
    object oItem = GetFirstItemInInventory(oPC);
    // Start the loop. The next statement says to
    // stay in the loop until "oItem" no longer points

```


This need to be in the "Text Appears When" Script node of the Text line you want to appear to the PC if he/she has the correct amount of said Item. Change "ITEM_TAG" to that of the Item you are looking for and set nMarkCount to the number of items you want.

```
#include "nw_i0_plot"
int StartingConditional()
{
    //check inventory of oPC for oQuestItem and get iNumItems
    string sMark = "ITEM_TAG";
    int nMarkCount = 2;
    object oMark = GetObjectByTag(sMark);
    if(GetNumItems(GetPCSpeaker(),sMark) >= nMarkCount)
        return TRUE;
    return FALSE;
}
```

If you want to take the Items in the "Action Taken" node of the next Text node. We can use TakeNumItems().

Place this script in the "Actions Taken" of the next NPC response. Again change "ITEM_TAG" to that of the Item your looking for and set nMarkCount to the number of Items to take.

```
#include "nw_i0_plot"
void main()
{
    string sMark = "ITEM_TAG"; // Enter tag of your Item
    object oMark = GetObjectByTag(sMark);
    int nMarkCount = 2; // Change to num Items you want
    if(GetNumItems(GetPCSpeaker(),sMark) >= nMarkCount)
    {
        TakeNumItems(GetPCSpeaker(),sMark,nMarkCount);
        // Uncomment the following line to Give the PC an Item
    Also
        // CreateItemOnObject("item_ResRef",GetPCSpeaker(),1);
    }
}
```

Sample Conversation

```
+Root
-NPC: "I see you have 2 Items" [Text appears When script]
---PC: "Yes I do"
----NPC:"Thanks" [Actions Taken]
-NPC: "I need you to get 2 Items"
---PC: "Ok, Be right back"
```

How do I make sure the PCs have a certain Item in possession before they are allowed to rest?

In the Module properties under the "Events" tab is a script handle marked OnPlayerRest. This is where you can make scripts to do things to the PC when they start to rest, end

their Rest, or Cancel their rest.

This is a basic script that will check for an Item the PC must have to be able to rest. The Item in this example is FOOD. Change that to the TAG of the item you are looking for. Or you can eliminate / add to the list.

```
void main()
{
    object oPC = GetLastPCRested();
    object oFood = GetItemPossessedBy(oPC, "FOOD");
    if(GetLastRestEventType() == REST_EVENTTYPE_REST_STARTED)
    {
        if(!GetIsObjectValid(oFood))
        {
            // This will prevent him from resting
            AssignCommand(oPC, ClearAllActions());
            // Add more conditions here.
        }
    }
}
```

NOTE: Canceling the PCs rest in this fashion will also trigger the REST_EVENTTYPE_REST_CANCELLED

How do I show the Entire Map to the PC if they buy a map from a merchant like in the OC?

This actually quite simple. First you must make an Item to be used as a Map that the PC can buy / find. Give it a Tag something like "MAP" or "MapToSewers". In fact you can make one for each area if you want.

Place this script in the OnEnter of Each area you want the show the PC if they have a map. Change the word MAP to that of the Item TAG of your map for that area.

```
void main()
{
    object oPC = GetEnteringObject();
    // First lets make sure a PC entered
    if(GetIsPC(oPC))
    {
        // Check for the map
        if(GetIsObjectValid(GetItemPossessedBy(oPC, "MAP")))
        {
            ExploreAreaForPlayer(OBJECT_SELF, oPC);
        }
    }
}
```

For multiple areas and Maps you can use the Same script in the OnEnter of each area so you don't have 10 different map scripts. However the coding is a bit more complex. We need to make sure the PC has the map for the area they are currently in. In this script you will need to specify the Area TAG for each map check.

```

void main()
{
    object oPC = GetEnteringObject();
    // First lets make sure a PC entered
    if(GetIsPC(oPC))
    {
        // Check for the map A
        if(GetIsObjectValid(GetItemPossessedBy(oPC, "MAPA")))
        {

ExploreAreaForPlayer(GetObjectByTag("MAPA_Area_Tag_Hear"), oPC);
        }
        // Check for map B
        if(GetIsObjectValid(GetItemPossessedBy(oPC, "MAPB")))
        {

ExploreAreaForPlayer(GetObjectByTag("MAPB_Area_Tag_Hear"), oPC);
        }
        // Check for map C
        if(GetIsObjectValid(GetItemPossessedBy(oPC, "MAPC")))
        {

ExploreAreaForPlayer(GetObjectByTag("MAPC_Area_Tag_Hear"), oPC);
        }
    }
}

```

Of course there are other, more advanced ways to do multiple maps but if you only have a few, this is the easiest to understand and use.

Also Note: Do NOT tag your maps with the same tag as the area. Although it may work when you test it, chances are it won't work after adding more areas. The reason is how `GetObjectByTag()` works. It may get the area (which is what we want) or it may Get the Map item on the PC. which is what we don't want.

Finally, if you simply want to explore the area for a PC with no map, use this

```

void main()
{
    object oPC = GetEnteringObject();
    if(GetIsPC(oPC))
    {
        ExploreAreaForPlayer(OBJECT_SELF, oPC);
    }
}

```

How can I make it so that after the conversation the NPC will turn back and face a certain Direction

Doing this really require no scripting, it's already there for you. We just need to make it work.

Place your NPC and face him in the desired direction. Note the NPC TAG. I'll call my NPC tag, "NPC_TAG". Place a Post in the same location and also face that in the direction you want your NPC to face. Right Click on the Post and Select "Properties". Enter POST_ plus the tag of your NPC. So mine would look like this, POST_NPC_TAG.

Now open the NPC's conversation and click the last Tab in the bottom right called "Current File". In the End Conversation Normally and Aborted nodes you will see a script named nw_walk_wp. This script is the default for "WalkWaypoints". This script may not be attached to your module. If you don't see the "preview" of the script in the window below it, do the following if you have not already done so. (Note: you should only need to do this once per module) Select the nw_walk_wp and Click EDIT. Click some place in the script, hit space then backspace, then click save. The nw_walk_wp script should now appear in the preview window and it will be added to your scripts list on the left.

Now, after a Conversation the NPC will turn and face the Direction of his post, and he will also return to it after combat.

NOTE: This was a problem after patch 1.25 (I believe) when Bioware added this script so NPCs walking waypoints would continue to walk them after a Conversation. However as stated before, the script is not attached to the module unless that too has been fixed. Also note: The nw_walk_wp script will Clear the actions of a NPC. Any scripts attached to the "Action Taken" of an ending conversation node that affects the NPC may fail. If you are having problems getting a "Actions Taken" script to function, try removing the nw_walk_wp script from the End Conversation Node.

How do you use Waypoints with NPCs?

This is all done in the toolset for any basic Waypoint walking NPC.

- 1) Place your NPC where you want him/her/it to start at.
- 2) Right Click on the NPC and select "Create Waypoint".
- 3) Keeping the NPC Selected (Green box around it). Right click where you want the NPC to walk to first and select "Create Waypoint".
- 4) Right click again where you want him to move too next, select "Create Waypoint"
- 5) Continue doing this until the path you want the NPC to take is complete. Make sure the NPC is "Selected" the entire time you are adding waypoints.

Look in the Left side bar under Waypoints and you will see all the Waypoints you just added. Should look something like this.

WP_TagOfNPC_01

WP_TagOfNPC_02

WP_TagOfNPC_03

Ect.

NOTE: Make sure each NPC that is walking waypoints have a Unique Tag. If you Place

5 guards and they all have the same tag, lets say GUARD, then ALL 5 Guards will walk the SAME set of waypoints. Unless this is what you want them to do, Change the Tag of each one you want to walk a different set of Waypoints.

You Can Add Waypoints Manually by Following the Naming convention as shown above. Place down your Waypoint and Change the Tag to WP_TAGofNPC_01 incrementing the Number by 1 for each additional Waypoint.

This is handy for making NPCs Cross Transition boarders.

Example:

I place a guard in Area 1 and have the following waypoints. (Guard Tag is GUARD)

WP_GUARD_01

WP_GUARD_02

WP_GUARD_03

Now I want the Guard to continue into Area2. So in area 2 I place 3 waypoint and change each TAG of the Waypoint to the following:

WP_GUARD_04

WP_GUARD_05

WP_GUARD_06

Now my NPC guard will walk back and forth between Area1 and Area2.

Known Issue

If there are NO PCs in the area, NPCs become lazy and they don't want to continue walking there Waypoints. Sometimes the NPCs will leave Area1 into Area2 and not return until the PC enters Area2.

What function do "#include" or "<span..." do??

As per the Lexicon;

```
#include
```

This declaration is used to load any external code libraries that might be needed by a script (any other *.nss file). It needs to be placed above any code that uses the functions which will be loaded, but it is generally good practice to place #include directives at the top of a script before any code.

#include statements have only the file's name, surrounded by text qualifiers (double quote characters), without the file extension.

Since it is a compiler directive and not an actual code statement, a #include does not have a ending semicolon.

```
// top of script  
#include "NW_I0_GENERIC"
```

Previously compiled scripts that use the recompiled include will need to be recompiled as well before showing changes.

From the Toolset, Build > Build Module is a quick way to recompile multiple scripts in this situation.

Note: if you attempt to comment out a #include with a multiline comment ("/*" and "*/"), the compiler still attempts to include the specified file. Use the single line comment ("//") instead.

Basically, a #include tells the script to "Also look at" said script, or "include" it's contents with this script.

For example, look at this script:

```
int PCIsInArea(object oArea)
{
    object oOcup = GetFirstObjectInArea(oArea);
    while(GetIsObjectValid(oOcup))
    {
        if(GetIsPC(oOcup))
            return TRUE;
        oOcup = GetNextObjectInArea(oArea);
    }
    return FALSE;
}
void main()
{
    if(PCIsInArea(OBJECT_SELF))
    {
        // do something
    }
}
```

Notice this line **int PCIsInArea(object oArea)** At the top of the script. This is a custom function that I have made to use in my scripts. It is included with the void main() part of the script.

Instead of doing this each time I need that function, I can put the **int PCIsInArea(object oArea)** part into it's own file, usually with a bunch of other custom functions and then simply tell my void main() script to include it. Like this:

```
#include "my_function" // or what ever I name the file
void main()
{
    if(PCIsInArea(OBJECT_SELF))
    {
        // do something
    }
}
```

Now the function `PCIsInArea()` is available to this script.

How do you give each entering PC an Item as they enter the Module for the First time?

You can place a script in the `OnClientEnter` of the Module. This will trigger each time a PC enters the game and NOT trigger each time a NPC spawns in. This is how it is different from a Normal Area or Trigger `OnEnter`.

Place the Following script in the `OnClientEnter` of the Module. It will check each entering PC for the Item first. If they don't have one, it gives them one.

```
void main()
{
    object oPC = GetEnteringObject();
    if(!GetIsObjectValid(GetItemPossessedBy(oPC, "TAG_OF_ITEM"))
    && GetIsPC(oPC))
    {
        CreateItemOnObject("res_ref_of_Item", oPC, 1);
    }
}
```

You will need to add your Items TAG and RESREF in the appropriate places.

Can Waypoints Run Scripts?

Waypoints themselves can't run scripts, no.

It's probably easiest to simply paint down a trigger and use the `OnEnter` to run a script.

However, if you look in the `nw_i0_generic` Include file, Under the `WalkWaypoints` function. You will see a Line like this, That has been commented out.

```
//ActionDoCommand( SignalEvent( OBJECT_SELF, EventUserDefined(2)));
```

I haven't tried it yet but I'm sure if you Uncomment that line, it will fire a UD event on the NPC walking at each waypoint.

I have a script that Adds an Effect to a PC. How do I remove it?

Removing effects is difficult if not done properly. Not only must the Effects be *applies* properly, but the must also be removed properly. To remove an effect you must loop through all the effects on a PC, find the one you want, and then remove it.

(See Above “Adding and Removing Effects” for more info)

What does "!" mean in a Script?

! has a few different names... bang operator, inverse operator, exclamation-point function thingy, etc. Its general purpose is to call the opposite of whatever it is you're doing.

GetIsObjectValid(oTarget) will return TRUE if oTarget is a valid object.

!GetIsObjectValid(oTarget) will return TRUE if oTarget is (b)not a valid object.

Likewise "if (GetLocalInt(oTarget, sName) == 2)" will only fire if the int is set to 2. "if

(GetLocalInt(oTarget, sName) != 2)" will only fire if the int is set to anything but 2.

What's the difference between sDest and oidDest?

(The line in question follows: object oidDest = GetObjectByTag(sDest);)

sDest has been defined as a String, denoted by the "s".

oidDest has been defined as a Object.

object oidDest = GetObjectByTag(sDest); is looking for an Object, declared as oidDest with the matching TAG of sDest.

object is the function type. The function will return a Object or INVALID.

oiDest is what you are naming the return value. This could be anything you want to call it. it could very well be oHenry if you want.

= equals, not to be confused with the Boolean operator of "==" which mean IS equal TO.

GetObjectByTag(sDest) Function. Returns a object with the specified tag. If there is more than on object with the same tag, the function will "get" the first on it comes to starting in the last area added to the Module. See "GetObjectByTag()" in the Lexicon for more information.

sDest is a string variable defined somewhere in the script. Usually towards the Top. Lets

say sDest is equal to "Fred". So the line could have been GetObjectByTag("Fred");

; End of function call. This tells the compiler that your done with that line and it moves to the next. Keep in mind that loops (for, while, do, ect) Don't have a ; at the end.

How can I make my NPC sit Cross-legged and NOT get up when talked to?

The Easy way

Add this Line at the Bottom of the NPC OnSpawn script.

```
ActionPlayAnimation( ANIMATION_LOOPING_SIT_CROSS, 1.0, 10000.0f);
```

And add this Script to the First NPC Text line in the "Actions Taken" Script node.

```
void main()  
{  
  
    ActionPlayAnimation(ANIMATION_LOOPING_SIT_CROSS,1.0,10000.0f);  
}
```


Make sure all the "Other actions" are on Default. If the NPC has a Post or Waypoint, remove the nw_walk_wp script from his "End Conversation Normally / aborted" script nodes.

To make him stand up again, simply issue him a ClearAllActions();

Side Effects: Sometimes, depending on when heartbeat hits and the PC clicks the NPC, the NPC might stand and then sit right back down. Not a prob for most cases.

The more complex but solid way

Create a Invis Object and give it the same name and Portrait of the NPC. Place it right next to the NPC. Lets give it a Tag of "Talker" and MOST IMPORTANT, make it NON-Static. Uncheck the "static" box and leave "usable" UNCHECKED.

Attach the NPC's Conversation to the Object in the Advanced tab in the "Conversation" box. Remove the Convo from the NPC.

As before, Add this line to the NPC's OnSpawn script

ActionPlayAnimation(ANIMATION_LOOPING_SIT_CROSS, 1.0, 10000.0f);

(Do Not use place the script from above in the actions taken of the Convo.)

Replace the NPC's OnConversation Script with this script

```
///:////////////////////////////////////  
///: Created By: Preston Watamaniuk  
///: Altered by Jassper  
///: Created On: Oct 24, 2001  
///:////////////////////////////////////  
///: Keeps the NPC sitting down during a convo  
////////////////////////////////////  
#include "NW_I0_GENERIC"  
void main()  
{  
    int nMatch = GetListenPatternNumber();  
    object oShouter = GetLastSpeaker();  
    object oIntruder;  
    object oSpeaker =  
GetNearestCreature(CREATURE_TYPE_PLAYER_CHAR, PLAYER_CHAR_IS_PC);  
    if (nMatch == -1 && GetCommandable(OBJECT_SELF))  
    {  
        //ClearAllActions();  
        //BeginConversation();  
  
AssignCommand(GetObjectByTag("Talker"), ActionStartConversation(oSpeaker));  
        SetFacingPoint(GetPosition(oSpeaker));  
    }  
    else  
        if(nMatch != -1 && GetIsObjectValid(oShouter) &&  
!GetIsPC(oShouter) && GetIsFriend(oShouter))  
    {
```

```

        if(nMatch == 4)
        {
            oIntruder = GetLocalObject(oShouter,
"NW_BLOCKER_INTRUDER");
        }
        else if (nMatch == 5)
        {
            oIntruder = GetLastHostileActor(oShouter);
            if(!GetIsObjectValid(oIntruder))
            {
                oIntruder = GetAttemptedAttackTarget();
                if(!GetIsObjectValid(oIntruder))
                {
                    oIntruder = GetAttemptedSpellTarget();
                    if(!GetIsObjectValid(oIntruder))
                    {
                        oIntruder = OBJECT_INVALID;
                    }
                }
            }
        }
        RespondToShout(oShouter, nMatch, oIntruder);
    }
    if(GetSpawnInCondition(NW_FLAG_ON_DIALOGUE_EVENT))
    {
        SignalEvent(OBJECT_SELF, EventUserDefined(1004));
    }
}

```

Now the NPC should stay on his rump and the Convo is actually Triggered from the Invis Object.

NOTE: any Actions or other scripts in the Convo Effecting the NPC will no longer work. They will need to AssignCommand(oNPC... The Invis Object in now the OWNER or OBJECT_SELF of the Convo.

As before, if you wish the NPC to stand, issue a ClearAllActions() to him/her.

How does the OnHeartbeat Function?

OnHeartbeat functions will fire every 6 seconds no matter what. Even on areas that players are not in. And even on object, creatures, or NPC's in areas that players are not in.

It is believed that OnHeartbeats on Areas that players are not in may only fire about every 20 seconds or so

I have a script in the "Actions Taken" at the end of a Conversation, but when the conversation end the NPC just stands there as if the script is not triggered.

There are 2 things that can cause this. If you Conversation ends on a NPC text node, and the "Actions Taken" script is on that Text node the script probably won't fire. Always include a PC response to any NPC text node containing a "Actions Taken" script. The PC response can be as simple as "Ok" or "End Dialog".

The other thing that it can be is the Bioware script "nw_walk_wp". This script is meant to make a NPC return to, or re-start the WayPoint walk. However, this function also makes a call to ClearAllActions();. So any ending script in the "Actions Taken" will be cleared by the WalkWaypoint Call. Be sure to remove the nw_walk_wp script from the "End Conversation Normally / Aborted" script handles, they can be found in the "Current File" tab of the Conversation editor. If the NPC needs to return to walking waypoints, then make sure all the NPCs actions are complete Before the Convo ends.

What is a switch/case ?

A switch/case is a simplified "if" type of statement. Best utilized when checking a lot of conditions. For example, I could have,

```
int nRan = Random(10); //returns a random number between 0 and 9
if(nRan == 0)
{
    //Do stuff
}
if(nRan == 1)
{
    //Do stuff
}
if(nRan == 2)
{
    //Do stuff
}
if(nRan == 3)
{
    //Do stuff
}
if(nRan == 4)
{
    //Do stuff
}
if(nRan == 5)
{
    //Do stuff
}
if(nRan == 6)
{
    //Do stuff
}
if(nRan == 7)
{
    //Do stuff
}
if(nRan == 8)
```

```

    {
    //Do stuff
    }
if(nRan == 9)
{
//Do stuff
}

```

Or I could replace all that with a switch/case, like this,

```

switch (Random(10))
{
case 0:
//Do stuff
break;
case 1:
//Do Stuff
break;
case 2:
//Do stuff
break;
case 3:
//Do Stuff
break;
case 4:
//Do stuff
break;
case 5:
//Do Stuff
break;
case 6:
//Do stuff
break;
case 7:
//Do Stuff
break;
case 8:
//Do stuff
break;
case 9:
//Do Stuff
break;

```

One thing to remember is that you can't define variables inside a case statement unless you use curly brackets. {}

```

switch (Random(10))
{
case 0:
{
//Define variables
break;
}
case 1:
{
//Define variables
break;
}

```

```
}
```

A switch/case is best used when each "case" is short, like setting an int.

```
int nTalent
switch (Random(10))
{
  case 0:
    nTalent = 5;
    break;
  case 1:
    nTalent = 10;
    break;
//ect
```

Keep in mind that this eliminates the use of the "else" statement. As soon as one of the case statements return true, the code in that case is executed and the switch exits and the rest of the script is executed.

The break; statement forces the switch to exit. You can leave the break statement out if you want but then the code will run down the entire list of case even after finding a case statement that is TRUE.

For example, lets say int i = 3. If we build a switch/case like this:

```
switch(i)
{
  case 1:
    //this will not execute
  case 2:
    //this will not execute
  case 3:
    //this WILL execute
  case 4:
    //this WILL execute
}
```

case 4 will execute because case 3 returned TRUE, and without a break; statement the switch assumes all others after it are also TRUE. (basically speaking) Sometimes this can be used as a desired effect.

The case number that returns TRUE and is executed is the number that is equal to the switch result. So if you count by Tens, your case numbers would also count by tens. For example, lets say int i can equal either 10, 20, or 30. Our switch/case would look like this.

```
switch (i)
{
  case 10:
    //Do stuff
    break;
  case 20:
    //Do stuff
    break;
```

```
case 30:  
  //Do stuff  
break:  
}
```

I just re-installed NWN. Do I have to re-play all the chapters again to unlock them?

AND

How can I open all the chapters in my Toolset without having to play them all?

Go to C:\NeverwinterNights\nwn\ and open the file named "nwnplayer.ini" notepad. Under the Heading [Game Option] add this line *exactly* **CODEWORD=hacktastic**. That will unlock all chapters of the OC.

To open them in the Toolset. Select "Open Existing module". At the bottom of the Module list screen click on the button "Campaign Modules" All chapters should now be listed in the box above.

What's the Basics for making a Merchant?

The Basics of it is fairly simple now with the addition of the Store Setup Wizard, no scripting required.

- 1) Select and place a NPC in your Module. Right click on the NPC and select "Setup Store". The Store Setup Wizard should appear.
- 2) The wizard gives you a NPC text line and 2 possible PC responses. The First line is what the NPC says when a player Clicks to talk to the NPC. You can edit it here if you like. The Next 2 lines are the players 2 options. Either "Yes. I'd like to see what you have for sale." or "No, thanks." Both of these can be edited here also. NOTE: you can still edit the conversation normally from the conversation editor.
- 3) At the bottom of the first screen you have 2 places for files names. You can use the ones that the Wizard gives you or edit them now. The first file name is for the Shop keepers Conversation file. You may want to re-name it now so you can Identify it later if need be. The second file name is for the name of the script that opens the store when the player selects "Yes. I'd like to see what you have for sale." You may also want to indicate what NPC merchant this is for.
- 4) Click "Next".
- 5) Now you need to choose a store. this can be done using the "Pre-made" stores or one of your own design. For this example I will select General Store from the standard list. The "New" button will create a new empty store in the custom list, that you can add what items you want to. The "Edit Copy" button will make a Copy of the standard store you

select. This too will be placed in the Custom list and you can add to or take things out of that store.

6) After you have chosen your store, click "Next". and then "Finish". You will notice a Waypoint placed where the NPC is standing and it will also appear under "Merchants" in the Left side bar.

That's *basically* all there is to it.

Who do I make my NPC say something while in combat?

1 - Uncomment the line, SetSpawnInCondition(NW_FLAG_SPECIAL_COMBAT_CONVERSATION); in the NPC's OnSpawn Script, save it with a new name.

2 - Make a Conversation file for the NPC. If the NPC already has one, then add a Text line as the *very first* thing the NPC says. Make this as a on-liner with no response.

4 - In the "Text Appears When" script node for that new text Line, drop down the Script menu and select this script, nw_d2_gen_combat

Now when in combat, the NPC will speak that line.

How could I easily add multiple journal entries upon the player entering the module?

Adding journal entries is simple and can be done in any scripting node. The Script command is AddJournalQuestEntry() which can be added to any script or used in a script you create. To add Multiple journal entries you would simply have multiple AddJournalQuestEntry() lines in the script.

Here is an example on how to add Journal entries when a PC enters the Module. Note that there is not a check to see if the PC already has this entry. It is really not needed as the entry will not add a second time if the PC already has it.

```
void main()
{
    object oPC = GetEnteringObject();
    if(GetIsPC(oPC))
    {
        AddJournalQuestEntry("Quest_1",10,oPC);
        AddJournalQuestEntry("Quest_2",10,oPC);
    }
}
```

In the script, replace "Quest_1" and "Quest_2" with the name(s) of your Journal entries (include the quotes). The Number 10 is the Entry ID number. Change that to which ID of the Entry you want the PC to have. You can include as many AddJournalQuestEntry()'s

as you need.

There is a bit more to that command. I suggest looking it over in the Lexicon. The script above can be modified to *override* any older quest entries and the script applied to a trigger or conversation

(More Info NWNLexicon:

<http://www.reapers.org/nwn/reference/compiled/function.AddJournalQuestEntry.html>)

How do I know what Journal Entry the PC has?

There is a variable set each time a PC gets a entry update. This can be checked by adding the *entry ID number* to the constant NW_JOURNAL_ENTRY.

In the Example script, I check to see if the PC has the Entry Quest_1 with the ID number 25

```
void main()
{
    object oPC = GetEnteringObject();
    string sQuestToken = "Quest_1";
    if(GetLocalInt(oPC, "NW_JOURNAL_ENTRY" + sQuestToken) == 25)
    {
        //Do whatever
    }
}
```

The String sQuestToken can be changed to your Journal Entry name. The next script is an example on how to find out what there last journal entry was.

```
void main()
{
    object oPC = GetPCSpeaker();
    string sQuestToken = "Quest_1";
    int nEntry = GetLocalInt(oPC, "NW_JOURNAL_ENTRY" +
sQuestToken);
    //Add coding here
}
```

nEntry will be equal to the last ID number added.

This type of thing would be handy during conversations to determine what Text node to display, or what quest Entry ID to add next.

What's the fastest way to tie the respawn of a PC to a specific way-point?

Simple. Add a Waypoint with the Tag "NW_DEATH_TEMPLE" in the location you want the PCs to respawn, no scripting changes needed.

How do I make a lever actually do something?

This is a custom function, that when placed in the OnUsed of a lever, will play it's animation and return a TRUE (ON) or FALSE (OFF) state to the script. You can either add it to a include file, or simply add it to the top of your void main() script as in the example.

When using a lever, be sure to set it's "Initial State" to "Deactivated"

```
////////////////////////////////////
// Place the Lever and set it State to "Deactivated"
// Use in the OnUsed of a Lever or switch
////////////////////////////////////
int FlipSwitch(object oLever = OBJECT_SELF)
{
    //lever animation
    if (GetLocalInt(OBJECT_SELF,"nToggle") == 0)
    {
        PlayAnimation(ANIMATION_PLACEABLE_ACTIVATE,1.0,1.0);
        SetLocalInt(OBJECT_SELF,"nToggle",1);//set "ON"
        return TRUE;
    }
    else
    {
        PlayAnimation(ANIMATION_PLACEABLE_DEACTIVATE,1.0,1.0);
        SetLocalInt(OBJECT_SELF,"nToggle",0);//set "OFF"
        return FALSE;
    }
    //end level animation
}
void main()
{
    if(FlipSwitch())
    {
        // Add code for ON state
    }
    else
    {
        // Add code for OFF state, if you wish
    }
}
}
```

How do I strip a PC of all Items and Gold?

This question has been asked more times than I can count. This is the basics of how to strip a PC using the OnEnter of a trigger, area, or OnClientEnter of a Module. Further modification can be done for your application.

```
void main()
{
    object oPC = GetEnteringObject();
    if(!GetIsPC(oPC)) return;
    // Check Equip Items and get rid of them
    int i;
```

```

        for(i=0; i<14; i++)
        {
            object oEquip = GetItemInSlot(i,oPC);
            if(GetIsObjectValid(oEquip))
            {
                SetPlotFlag(oEquip,FALSE);
                DestroyObject(oEquip);
            }
        }
// Check general Inventory and clear it out.
    object oItem = GetFirstItemInInventory(oPC);
    while(GetIsObjectValid(oItem))
    {
        SetPlotFlag(oItem,FALSE);
        DestroyObject(oItem);
        oItem = GetNextItemInInventory(oPC);
    }
//Take their Gold
    int nAmount = GetGold(oPC);
    if(nAmount > 0)
    {
        AssignCommand(oPC,TakeGoldFromCreature(nAmount,oPC,TRUE));
    }
}

```

How do I make a placable start a conversation with a PC on Use?

Make a conversation like normal and attach it to the Placeables in the Placeables Properties > Advanced tab. Add the following script to the Placeables OnUsed script node.

```

void main()
{
    ActionStartConversation(GetLastUsedBy());
}

```

How do I make a NPC "flee" the area after a conversation?

Place a Waypoint located where you want the NPC to move to and vanish. Give it a Unique Tag. Note: there is a default tag called "NW_EXIT". however, if you use this a lot, the NPC could get confused and move to the wrong one.

In the Conversation Editor, Click on the last TAB in the lower right "Current File". Place this script in the "End Conversation Normally" script handle. You can also place it in the "Aborted" handle if you like.

```

#include "nw_i0_plot"
void main()
{
    // Change FALSE to TRUE if you want the NPC to Run.
    // If you use the default Waypoint Tag of "NW_EXIT"
    // then all you need is EscapeArea();
}

```

```

    // Or, EscapeArea(TRUE); if you want him to Run.
    EscapeArea(FALSE, "Waypoint_Tag");
}

```

How do I make Floaty Text, or Speech on objects and doors?

For Objects, you would place this script in the OnUsed handle. Remove the parts (or comment out) you don't want to use.

```

void main()
{
    object oPC = GetLastUsedBy();
    // For Floaty Text
    FloatingTextStringOnCreature("Your Message",oPC);
    // For Speaking a string, Text is displayed for one
    Heartbeat.
    SpeakString("Your Message");
    // For a Privet message
    SendMessageToPC(oPC,"Your Message");
}

```

For a Door, you can place it in the OnUsed, but usually doors that say something usually is because it is locked. In that case you would need to place *this* script in the OnFailed to open. Same as above but oPC is now GetClickingObject()

```

void main()
{
    object oPC = GetClickingObject();
    // For Floaty Text
    FloatingTextStringOnCreature("Your Message",oPC);
    // For Speaking a string, Text is displayed for one
    Heartbeat.
    SpeakString("Your Message");
    // For a Privet message
    SendMessageToPC(oPC,"Your Message");
    // To start a Conversation
    ActionStartConversation(oPC);
}

```

How can I make a "Level up" Script?

This is a long over-due question that needs answered. There are several ways to do this, I will post 2 simple scripts here that can be modified to your own needs.

The basic formula for gaining a level is this,

$((\text{CurrentLevel} + 1) * \text{CurrentLevel}) / 2 * 1000$

i.e. To Gain 5th level from 4th level the PC would need,

$((4 + 1) * 4) / 2 * 1000 = 10,000$ XP points needed for 5th level.

Here is a Script that will allow you to level up 1 level each time an object is used.

```

void main()

```

```

{
    object oPC = GetLastUsedBy();
    int nCurrentL = GetHitDice(oPC);
    int nXP = ((nCurrentL + 1)*nCurrentL)/2)*1000;
    SetXP(oPC,nXP);
}

```

The next script is a script that will set your level to 5 as long as you are already less than level 5. You can change the int nLevelMax to what ever you want the max level to be.

```

void main()
{
    object oPC = GetLastUsedBy();
    int nCurrentL = GetHitDice(oPC);
    int nLevelMax = 5;
    if(nCurrentL < nLevelMax)
    {
        int nXP = ((nLevelMax*(nLevelMax - 1))/2)*1000;
        SetXP(oPC,nXP);
    }
}

```

Again this is in a OnUsed of an object. If you would like to use it in a conversation then change this line,

object oPC = GetLastUsedBy();

to this,

object oPC = GetPCSpeaker();

and place it in a "Actions Taken" script node of your conversation.

How do I use the Special Conversations mentioned in the OnSpawn Script?

First, build the NPCs conversation using these guild lines.

```

+Root
[line1]NPC - "Say my greeting when I perceive a PC"
[line2]NPC - "Say my Battle cry if I'm attacked!"
[line3]NPC - "Start my normal conversation here."

```

NOTE: Line 1 and Line 2 CAN NOT have a PC response on them.

In the convo editor, select line one. In the "Text Appears When" script box, drop down the menu and select the script named "nw_d2_gen_check". Click on line 2 and in the same "Text Appears When" script node, select the script named "nw_d2_gen_combat". Make sure Line 1 is at the top, and line 2 is just underneath. Save the convo.

Now, open the properties for the NPC. Go to the Scripts TAB and click "Edit" in the OnSpawn script. Uncomment (remove the 2 "//") these 2 lines,
SetSpawnInCondition(NW_FLAG_SPECIAL_CONVERSATION);
SetSpawnInCondition(NW_FLAG_SPECIAL_COMBAT_CONVERSATION);

Resave the script with a new name and be sure it is attached to the NPCs OnSpawn script node.

The NPC should say his greeting (line 1) when he sees a PC. He will repeat this line each time a PC is perceived. If he is attacked, he should say his combat saying (line 2).

NOTE: If the NPC is already a Hostile creature, Line 1 should be a battle cry, something like "ATTACK!" or "CHARGE!" as he will only say this once OnPerception. Line 2 will repeat *during* combat.

How do I make my NPC turn and face there previous facing after being clicked on?

Simple. Place and face your NPC however you want to. Next, in the same spot, place a Waypoint and Tag it "POST_tagofNPC". So if the Tag of the NPC is "GUARD" your waypoint tag would be "POST_GUARD". Point the waypoint arrow in the direction you want the NPC to face.

Now create a conversation for the NPC. This conversation can be a One-Liner, full blow convo, or left blank. While in the Convo editor, click the "Current File" tab and verify the file "nw_walk_wp" script is listed in the "End Conversation Normally and Aborted" script nodes. If not, pull down the menu and select it. Now click "Edit" to open the file. Click anywhere in the script, hit the space bar, then back-space. Click "Save". This will attach the script to your module. Don't ask me why you have to do this, I just know that you do. (sometimes anyways).

Now, in game when you click on the NPC, they will face you then turn back and face the direction of the POST after the conversation.

How do I make my NPCs motionless?

There are several options here, I'll just mention a few.

1 - You can use ApplyEffect() in the NPCs OnSPawn and apply the EffectParalyze() effect.

Or

2 - You can do the same as above with a Blank convo or no more than a On-Liner. This usually makes him turn fast enough that it is hardly noticed.

3 - You can also add this to the NPCs OnConversation script.

```
object oFace = GetWaypointByTag("POST_TAGofNPC");  
SetFacing(GetFacing(oFace));
```

Place the same type of waypoint as mentioned above, with a tag of "POST_TagOfNPC".

There are several other way also, feel free to play with it.

How can I restrict how often PC's can rest?

This script will allow you to set how many minutes of real time must pass between rests. Note that real time minutes and game minutes are actually the same, and the same for seconds. The difference comes in when going from minutes to hours. This script takes this into account with the HoursToSeconds function:

```
// By Orion Light
// Gets the current time in minutes as an integer
int GetTotalTimeMinutes()
{
int nTimeMinutes; // Declaring a variable
int nMinutes = GetTimeMinute(); // get number of minutes
int nHours = GetTimeHour(); // get hours
int nDays = GetCalendarDay(); // Get days
int nMonths = GetCalendarMonth(); // Get months
int nYears = GetCalendarYear(); // Get years
float fMinutesPerHour = HoursToSeconds(1); // Get number of
seconds in 1 hour
int nMinutesPerHour = FloatToInt(fMinutesPerHour); // Change to
int
nMinutesPerHour /= 60; // Divide by 60 to get minutes in an hour
// If you don't do this, any variance in the module hours/minutes
can mess it up
nHours *= nMinutesPerHour; // Convert hours into minutes
nDays *= (nMinutesPerHour * 24); // Convert days into minutes
nMonths *= (nMinutesPerHour * 672); // Convert months into minutes
nYears *= (nMinutesPerHour * 8064); // Convert years into minutes
nTimeMinutes = (nMinutes + nHours + nDays + nMonths + nYears); //
Add all together
return nTimeMinutes; // The return of the function is the total
number of minutes
}
// Does not allow rest until nLimit minutes have passed
// Cancelled rest counts as rest and will reset the timer
void main()
{
int nLimit = 2; // SET NUMBER OF MINUTES ALLOWED HERE
object oPC = GetLastPCRested(); // Gets the PC
int nRestType = GetLastRestEventType(); // Get the last rest event
type
int nTotalTimeMinutes = GetTotalTimeMinutes(); // Record the
current time
int nLastRestTimeMinutes = GetLocalInt(oPC, "lastrested"); //
Check when PC last rested
int nTimeElapsed = (nTotalTimeMinutes - nLastRestTimeMinutes); //
Get the difference
int nTimeLeft = (nLimit - nTimeElapsed); // Calculate how long
left untill rest allowed
if (nRestType == REST_EVENTTYPE_REST_STARTED) // If rest just
started
{
if (nTimeElapsed < nLimit) // And elapsed time is less than
the limit allowed
```

```

        {
            AssignCommand(oPC, ClearAllActions()); // Cancel the rest
            SendMessageToPC(oPC, "You may not rest for " +
                IntToString(nTimeLeft) + " minutes");
            // Sends the PC a message
        }
    }
    if ((nRestType = REST_EVENTTYPE_REST_FINISHED) && (nTimeElapsed
        >= nLimit))
        // If rest is finished and time elapsed is greater than time
        allowed
        // Prevents the timer resetting if rest was cancelled because of
        the limit
        {
            SetLocalInt(oPC, "lastrested", GetTotalTimeMinutes());
            // Set the time PC rested as a local int
        }
    }
}

```

There are simpler ways to do it, but this method allows feedback on how long it will be until being allowed to rest.

How do I combine several scripts to activate on the same event, such as the OnHeartbeat?

The frustrating way is to copy and past all the scripts in to one and pray just before hitting the "compile" button. Chances are, they won't compile for several reasons.

I found it much easier to do it this way instead. I'll use the OnHeartbeat handle for example and lets say I have 3 scripts I want to combine.

1) Save each script with there own name. i.e. "heartbeat_01", "heartbeat_02", "heartbeat_03" or whatever you desire.

2) In the Actual OnHeartbeat you want all of these to run, write a new script like this one.

```

void main()
{
    ExecuteScript("heartbeat_01", OBJECT_SELF);
    ExecuteScript("heartbeat_02", OBJECT_SELF);
    ExecuteScript("heartbeat_03", OBJECT_SELF);
}

```

That's all there is to it! Now they each run in the OnHeartbeat of the object / area / or module they are attached to without having to "re-script" to get them to compile.

Is there a way to define which immobile animations you want an NPC to do?

You can use the ActionPlayAnimation(int, float, float) function.

To follow your example, you could add this to the Performers OnHeartbeat script,

```
ActionPlayAnimation(ANIMATION_FIREFORGET_GREETING);
ActionPlayAnimation(ANIMATION_LOOPING_GET_LOW,1.0,1.0);
ActionPlayAnimation(ANIMATION_LOOPING_GET_MID,1.0,1.0);
ActionPlayAnimation(ANIMATION_LOOPING_GET_LOW,1.0,1.0);
ActionPlayAnimation(ANIMATION_FIREFORGET_TAUNT);
ActionPlayAnimation(ANIMATION_FIREFORGET_BOW);
```

This is probably the most you would get into a OnHeartbeat, If it loops before the previous OnHeartbeat actions are complete, you will soon see a lot of lag.

Note the ANIMATION_FIREFORGET* types don't need a "duration" time because it is not used for those types. the Looping Animations need one to so the NPC knows *how long* they should do the animation for.

How do I make a sign use a floaty for its name?

Here's a script my husband wrote for me that might be useful: Dynamic Sign Script, to make it so that each sign, when used, would have a (short) floaty-text-string above it, unique for each sign, with only one script.

```
///://////////////////////////////////////////
///: s_sign_onuse
///: Dynamic Merchant Sign Script
///: Copyright (c) 2002 David Lynch
///://////////////////////////////////////////
void main()
{
    int nStart = 0;
    string sPrint = "\n";
    string sName = GetTag(OBJECT_SELF);
    string sTemp = GetSubString(sName, nStart, 1);
    while (sTemp != "") {
        if (sTemp == "_") {
            if (GetSubString(sName, (nStart + 1), 1) == "_") {
                sPrint = sPrint + " ";
                nStart++;
            } else sPrint = sPrint + " ";
        } else sPrint = sPrint + sTemp;
        nStart++;
        sTemp = GetSubString(sName, nStart, 1);
    }
    SpeakString(sPrint);
}
```


Make your sign "Plot" (if desired) and "Usable".

Make the NAME of your sign something generic, like "Merchant Sign".

Make the TAG of your sign be whatever text you want to float above the sign, with a single underscore signifying a space, and a double underscore signifying an apostrophe.

Eg: Having a sign with the tag: "Tom__s_General_Store" would read: "Tom's General Store" when a player uses that sign.

Your only limitation is that the TAG of an object only allows so many characters (I believe something like 32), but with most sign descriptions, you won't go over that number anyway.

Edit: Not limited to signs, this can be used for any placeable, but it's probably most common to be used for signs and other items players can use to get a description.

How do I make the corpse face up?

Get the Area to kill it.

Bizarre, huh. It seems that if you kill yourself, you are smacking yourself from behind, and end up face down. Whereas if the Area kills you, it is smacking you from the front, and you end up face up. Go figure.

For example, the creature's *OnSpawn* script:

```
void main()
{
    object oArea = GetArea( OBJECT_SELF )
    ;
    SetLocalObject( oArea, "oSubject", o )
    ;
    SignalEvent( oArea, EventUserDefined( 107 ) )
    ;
    return
    ;
}
```

The Area's *OnUserDefined* script:

```
void main()
{
    switch( GetUserDefinedEventNumber() )
    {
        case 107
            :
                // An AREA killing an OBJECT puts the OBJECT in a
                face-up position.
                ApplyEffectToObject( DURATION_TYPE_INSTANT,
                EffectDeath(), GetLocalObject( OBJECT_SELF, "oSubject" ) )
                ;
                break
            ;
    }
}
```



```

//:: Created On: November 6, 2001
//::////////////////////////////////////
void main()
{
    // AssignCommand(GetLastPlayerDying(), ClearAllActions());
    // AssignCommand(GetLastPlayerDying(), SpeakString( "I
Dying"));
    //
PopUpGUIPanel(GetLastPlayerDying(), GUI_PANEL_PLAYER_DEATH);
// * April 14 2002: Hiding the death part from player
effect eDeath = EffectDeath(FALSE, FALSE);
    ApplyEffectToObject(DURATION_TYPE_INSTANT, eDeath,
GetLastPlayerDying());
}

```

The 3 commented out lines you see I believe are debug lines that bio left in. They are not needed. The only lines that matter here are the effect and ApplyEffect lines. This is what actually kills the PC, **not** the creature they were fighting at the time.

So once your dead, the OnPlayerDeath script fires.

```

//::////////////////////////////////////
//:: Death Script
//:: NW_O0_DEATH.NSS
//:: Copyright (c) 2001 Bioware Corp.
//::////////////////////////////////////
/*
    This script handles the default behavior
    that occurs when a player dies.
    BK: October 8 2002: Overridden for Expansion
*/
//::////////////////////////////////////
//:: Created By: Brent Knowles
//:: Created On: November 6, 2001
//::////////////////////////////////////
/*
void ClearAllFactionMembers(object oMember, object oPlayer)
{
//    AssignCommand(oMember, SpeakString("here"));
    AdjustReputation(oPlayer, oMember, 100);
    SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // *
Player bad
    object oClear = GetFirstFactionMember(oMember, FALSE);
    while (GetIsObjectValid(oClear) == TRUE)
    {
        ClearPersonalReputation(oPlayer, oClear);
        oClear = GetNextFactionMember(oMember, FALSE);
    }
}
*/
void Raise(object oPlayer)
{
    effect eVisual = EffectVisualEffect(VFX_IMP_RESTORATION);
    effect eBad = GetFirstEffect(oPlayer);

```

```

ApplyEffectToObject(DURATION_TYPE_INSTANT,EffectResurrection(),oP
layer);

ApplyEffectToObject(DURATION_TYPE_INSTANT,EffectHeal(GetMaxHitPoi
nts(oPlayer)),
oPlayer);
    //Search for negative effects
    while(GetIsEffectValid(eBad))
    {
        if (GetEffectType(eBad) ==
EFFECT_TYPE_ABILITY_DECREASE ||
            GetEffectType(eBad) == EFFECT_TYPE_AC_DECREASE ||
            GetEffectType(eBad) ==
EFFECT_TYPE_ATTACK_DECREASE ||
            GetEffectType(eBad) ==
EFFECT_TYPE_DAMAGE_DECREASE ||
            GetEffectType(eBad) ==
EFFECT_TYPE_DAMAGE_IMMUNITY_DECREASE ||
            GetEffectType(eBad) ==
EFFECT_TYPE_SAVING_THROW_DECREASE ||
            GetEffectType(eBad) ==
EFFECT_TYPE_SPELL_RESISTANCE_DECREASE ||
            GetEffectType(eBad) == EFFECT_TYPE_SKILL_DECREASE
||
            GetEffectType(eBad) == EFFECT_TYPE_BLINDNESS ||
            GetEffectType(eBad) == EFFECT_TYPE_DEAF ||
            GetEffectType(eBad) == EFFECT_TYPE_PARALYZE ||
            GetEffectType(eBad) == EFFECT_TYPE_NEGATIVELEVEL)
        {
            //Remove effect if it is negative.
            RemoveEffect(oPlayer, eBad);
        }
        eBad = GetNextEffect(oPlayer);
    }
    //Fire cast spell at event for the specified target
    SignalEvent(oPlayer, EventSpellCastAt(OBJECT_SELF,
SPELL_RESTORATION, FALSE));
    ApplyEffectToObject(DURATION_TYPE_INSTANT, eVisual,
oPlayer);
}
void main()
{
    object oPlayer = GetLastPlayerDied();
    // * increment global tracking number of times that I died
    SetLocalInt(oPlayer, "NW_L_PLAYER_DIED", GetLocalInt(oPlayer,
"NW_L_PLAYER_DIED") + 1);
    // * BK: Automation Control. Autopcs ignore death
    if (GetLocalInt(oPlayer, "NW_L_AUTOMATION") == 10)
    {
        Raise(oPlayer);
        DelayCommand(1.0, ExecuteScript("crawl", OBJECT_SELF));
        return; // Raise and return
    }

    // * Handle Spirit of the Wood Death
    string sArea = GetTag(GetArea(oPlayer));
/*

```

```

        if (sArea == "MAP_M2Q2F2" &&
GetDistanceBetweenLocations(GetLocation(GetObjectByTag("M2Q2F2_M2
Q2G")),
GetLocation(oPlayer)) < 5.0 &&
GetLocalInt(GetModule(),"NW_M2Q2E_WoodsFreed") == 0)
    {
        int bValid;
        Raise(oPlayer);
        string sDestTag = "WP_M2Q2GtoM2Q2F";
        object oSpawnPoint = GetObjectByTag(sDestTag);

AssignCommand(oPlayer,JumpToLocation(GetLocation(oSpawnPoint)));
        return;
    }
*/
// * in last level of the Sourcestone, move the player to the
beginning of the area
// * May 16 2002: or the main area of the Snowglobe (to
prevent plot logic problems).
// * May 21 2002: or Castle Never
    if (sArea == "M4Q1D2" || sArea == "M3Q3C" || sArea ==
"MAP_M1Q6A")
    {
        //Raise(oPlayer);
        //string sDestTag = "M4QD07_ENTER";
        //object oSpawnPoint = GetObjectByTag(sDestTag);
//        AssignCommand(oPlayer, DelayCommand(1.0,
JumpToLocation(GetLocation(oSpawnPoint))));
// * MAY 2002: Just popup the YOU ARE DEAD panel at this point
        DelayCommand(2.5, PopUpDeathGUIPanel(oPlayer,FALSE, TRUE,
66487));
        return;
    }
// * make friendly to Each of the 3 common factions
AssignCommand(oPlayer, ClearAllActions());
// * Note: waiting for Sophia to make
SetStandardFactionReputation to clear all personal
reputation
    if (GetStandardFactionReputation(STANDARD_FACTION_COMMONER,
oPlayer) <= 10)
    {
        SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // *
Player bad
        SetStandardFactionReputation(STANDARD_FACTION_COMMONER,
80, oPlayer);
    }
    if (GetStandardFactionReputation(STANDARD_FACTION_MERCHANT,
oPlayer) <= 10)
    {
        SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // *
Player bad
        SetStandardFactionReputation(STANDARD_FACTION_MERCHANT,
80, oPlayer);
    }
    if (GetStandardFactionReputation(STANDARD_FACTION_DEFENDER,
oPlayer) <= 10)
    {
        SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // *
Player bad
        SetStandardFactionReputation(STANDARD_FACTION_DEFENDER,

```

```

80, oPlayer);
    }
    DelayCommand(2.5,
PopUpGUIPanel(oPlayer, GUI_PANEL_PLAYER_DEATH));
}

```

This script at first glance seems confusing. One thing to remember is that it has a lot of "left-over" stuff that only pertains to the Original campaign and really doesn't differ from the x0_o0_death script from the Hordes Campaign. If we take out all the garbage, we are left with this.

```

/*void ClearAllFactionMembers(object oMember, object oPlayer)
{
    // AssignCommand(oMember, SpeakString("here"));
    AdjustReputation(oPlayer, oMember, 100);
    SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // * Player bad
    object oClear = GetFirstFactionMember(oMember, FALSE);
    while (GetIsObjectValid(oClear) == TRUE)
    {
        ClearPersonalReputation(oPlayer, oClear);
        oClear = GetNextFactionMember(oMember, FALSE);
    }
}
*/
void Raise(object oPlayer)
{
    effect eVisual = EffectVisualEffect(VFX_IMP_RESTORATION);
    effect eBad = GetFirstEffect(oPlayer);

    ApplyEffectToObject(DURATION_TYPE_INSTANT, EffectResurrection(), oPlayer);

    ApplyEffectToObject(DURATION_TYPE_INSTANT, EffectHeal(GetMaxHitPoints(oPlayer)),
oPlayer);
    //Search for negative effects
    while(GetIsEffectValid(eBad))
    {
        if (GetEffectType(eBad) == EFFECT_TYPE_ABILITY_DECREASE ||
            GetEffectType(eBad) == EFFECT_TYPE_AC_DECREASE ||
            GetEffectType(eBad) == EFFECT_TYPE_ATTACK_DECREASE ||
            GetEffectType(eBad) == EFFECT_TYPE_DAMAGE_DECREASE ||
            GetEffectType(eBad) ==
EFFECT_TYPE_DAMAGE_IMMUNITY_DECREASE ||
            GetEffectType(eBad) == EFFECT_TYPE_SAVING_THROW_DECREASE
        ||
            GetEffectType(eBad) ==
EFFECT_TYPE_SPELL_RESISTANCE_DECREASE ||
            GetEffectType(eBad) == EFFECT_TYPE_SKILL_DECREASE ||
            GetEffectType(eBad) == EFFECT_TYPE_BLINDNESS ||
            GetEffectType(eBad) == EFFECT_TYPE_DEAF ||
            GetEffectType(eBad) == EFFECT_TYPE_PARALYZE ||
            GetEffectType(eBad) == EFFECT_TYPE_NEGATIVELEVEL)
        {
            //Remove effect if it is negative.
            RemoveEffect(oPlayer, eBad);
        }
        eBad = GetNextEffect(oPlayer);
    }
    //Fire cast spell at event for the specified target
    SignalEvent(oPlayer, EventSpellCastAt(OBJECT_SELF,
SPELL_RESTORATION, FALSE));
    ApplyEffectToObject(DURATION_TYPE_INSTANT, eVisual, oPlayer);
}
void main()
{
    object oPlayer = GetLastPlayerDied();

```

```

        // * increment global tracking number of times that I died
        SetLocalInt(oPlayer, "NW_L_PLAYER_DIED", GetLocalInt(oPlayer,
"NW_L_PLAYER_DIED") + 1);
        // * BK: Automation Control. Autopcs ignore death
        if (GetLocalInt(oPlayer, "NW_L_AUTOMATION") == 10)
        {
            Raise(oPlayer);
            DelayCommand(1.0, ExecuteScript("crawl", OBJECT_SELF));
            return; // Raise and return
        }
        // * make friendly to Each of the 3 common factions
        AssignCommand(oPlayer, ClearAllActions());
        // * Note: waiting for Sophia to make SetStandardFactionReputation to
clear all personal
reputation
        if (GetStandardFactionReputation(STANDARD_FACTION_COMMONER, oPlayer)
<= 10)
        {
            SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // * Player bad
            SetStandardFactionReputation(STANDARD_FACTION_COMMONER, 80,
oPlayer);
        }
        if (GetStandardFactionReputation(STANDARD_FACTION_MERCHANT, oPlayer)
<= 10)
        {
            SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // * Player bad
            SetStandardFactionReputation(STANDARD_FACTION_MERCHANT, 80,
oPlayer);
        }
        if (GetStandardFactionReputation(STANDARD_FACTION_DEFENDER, oPlayer)
<= 10)
        {
            SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // * Player bad
            SetStandardFactionReputation(STANDARD_FACTION_DEFENDER, 80,
oPlayer);
        }
        DelayCommand(2.5, PopUpGUIPanel(oPlayer, GUI_PANEL_PLAYER_DEATH));
    }
}

```

Note that the function "ClearAllFactionMembers()" is still commented out, but left in. This function, if used will clear the hostility of oMember towards oPlayer for custom Factions, just as it does for the standard factions. For example, if a PC attacks a faction that is not normally hostile to the PC, and the PC dies. When the Player respawns that faction will still be hostile towards them. This function can be used to clear that state so the faction is no longer hostile when the PC re-spawns. To demonstrate this we will need to alter the OnPlayerDying script as well. Remember that the Module kills the player in the OnPlayer Dying and to use the ClearAllFactionMembers() function we will need the object of the Players NPC killer. Functions like GetLastkiller() or GetLastHostileActor() are going to return the module object and not the NPC killer. More on this later.

The next function is the Raise() function. This is used to "Re-spawn" the PC if the Automation control is set on the Player object. Looking further down in the script, under the void main(), the first IF statement checks to see if the variable "NW_L_AUTOMATION" is set to 10, if so, then auto-respawn the PC. We can test that now if you like. Enter a Module and in the chat bar type **##DebugMode 1** and hit enter, then type **##dm_setvarint NW_L_AUTOMATION 10** and hit enter. The curser will change to a "targeter", click on your PC. Now attack something that you know can kill your PC. YOU WON'T DIE!

After the Rasie() function we come to the void main(). First the script gets the player that just died, then sets a variable int on that player tracking the number of times that player

died. So if you ever want to know how much a PC died during that game session, you can use `int iDied = GetLocalInt(oPlayer, "NW_L_PLAYER_DIED");` where `oPlayer` is the PC object in question. This was used in certain features of Hordes I believe. If you don't need it, you can safely take it out.

Next is the If statement we just discussed, it checks to see if the PC auto-respawn is turned on. This mainly comes in handy for testing. This too can be removed if not in use. Note that this block also runs a script called "crawl". This script does not exist so nothing is called.

Further down we see 3 if blocks these set "NW_G_Playerhasbeenbad" on the Player if they became hostile to either the Commoner, Defender, or Merchant factions then resets these factions to friendly. This is so if the player attacks a guard for example, and gets killed, the Player can re-spawn and not get re-attacked by the guards.

Then finally we have the delay to pop up the player GUI panel. This is the panel that gives the player a choice of "Load", "Respawn", "Exit". Load will allow the Player to re-load or load a different module. Exit takes the player back to the game startup screen and Respawn will, of course, run the `OnPlayerRespawn` script node.

Before we move on to the `OnPlayerRespawn` script, lets revisit the `ClearAllFactionMembers()` function and what must be done to make it work. This will also show how to "get" the object that killed the PC instead of the Module. The quickest way is to simply remove the `OnPlayerDying` script or comment out the `ApplyEffect` line. This will allow us to use `GetLastHostileActor()` in the `OnDeath` script. I should also mention here that doing so means that the player will become "unconscious" at 0 and not die until they reach -11 but there are ways around that as well. Now we can alter the `OnPlayerDeath` script like this,

```
void ClearAllFactionMembers(object oMember, object oPlayer)
{
    // AssignCommand(oMember, SpeakString("here"));
    AdjustReputation(oPlayer, oMember, 80);
    SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // * Player bad
    object oClear = GetFirstFactionMember(oMember, FALSE);
    while (GetIsObjectValid(oClear) == TRUE)
    {
        ClearPersonalReputation(oPlayer, oClear);
        oClear = GetNextFactionMember(oMember, FALSE);
    }
}

void Raise(object oPlayer)
{
    effect eVisual = EffectVisualEffect(VFX_IMP_RESTORATION);
    effect eBad = GetFirstEffect(oPlayer);
    ApplyEffectToObject(DURATION_TYPE_INSTANT, EffectResurrection(), oPlayer);

    ApplyEffectToObject(DURATION_TYPE_INSTANT, EffectHeal(GetMaxHitPoints(oPlayer)),
    oPlayer);
    //Search for negative effects
    while(GetIsEffectValid(eBad))
    {
        if (GetEffectType(eBad) == EFFECT_TYPE_ABILITY_DECREASE ||
            GetEffectType(eBad) == EFFECT_TYPE_AC_DECREASE ||
            GetEffectType(eBad) == EFFECT_TYPE_ATTACK_DECREASE ||
            GetEffectType(eBad) == EFFECT_TYPE_DAMAGE_DECREASE ||
```



```

        GetEffectType(eBad) == EFFECT_TYPE_DAMAGE_IMMUNITY_DECREASE ||
        GetEffectType(eBad) == EFFECT_TYPE_SAVING_THROW_DECREASE ||
        GetEffectType(eBad) == EFFECT_TYPE_SPELL_RESISTANCE_DECREASE ||
        GetEffectType(eBad) == EFFECT_TYPE_SKILL_DECREASE ||
        GetEffectType(eBad) == EFFECT_TYPE_BLINDNESS ||
        GetEffectType(eBad) == EFFECT_TYPE_DEAF ||
        GetEffectType(eBad) == EFFECT_TYPE_PARALYZE ||
        GetEffectType(eBad) == EFFECT_TYPE_NEGATIVELEVEL)
    {
        //Remove effect if it is negative.
        RemoveEffect(oPlayer, eBad);
    }
    eBad = GetNextEffect(oPlayer);
}
//Fire cast spell at event for the specified target
SignalEvent(oPlayer, EventSpellCastAt(OBJECT_SELF, SPELL_RESTORATION,
FALSE));
ApplyEffectToObject(DURATION_TYPE_INSTANT, eVisual, oPlayer);
}
void main()
{
    object oPlayer = GetLastPlayerDied();
    // * increment global tracking number of times that I died
    SetLocalInt(oPlayer, "NW_L_PLAYER_DIED", GetLocalInt(oPlayer,
"NW_L_PLAYER_DIED") + 1);
    // * BK: Automation Control. Autopcs ignore death
    if (GetLocalInt(oPlayer, "NW_L_AUTOMATION") == 10)
    {
        Raise(oPlayer);
        DelayCommand(1.0, ExecuteScript("crawl", OBJECT_SELF));
        return; // Raise and return
    }
    // * make friendly to Each of the 3 common factions
    AssignCommand(oPlayer, ClearAllActions());
    // * Note: waiting for Sophia to make SetStandardFactionReputation to clear
all personal
reputation
    if (GetStandardFactionReputation(STANDARD_FACTION_COMMONER, oPlayer) <= 10)
    {
        SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // * Player bad
        SetStandardFactionReputation(STANDARD_FACTION_COMMONER, 80, oPlayer);
    }
    if (GetStandardFactionReputation(STANDARD_FACTION_MERCHANT, oPlayer) <= 10)
    {
        SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // * Player bad
        SetStandardFactionReputation(STANDARD_FACTION_MERCHANT, 80, oPlayer);
    }
    if (GetStandardFactionReputation(STANDARD_FACTION_DEFENDER, oPlayer) <= 10)
    {
        SetLocalInt(oPlayer, "NW_G_Playerhasbeenbad", 10); // * Player bad
        SetStandardFactionReputation(STANDARD_FACTION_DEFENDER, 80, oPlayer);
    }
    object oKiller = GetLastHostileActor(oPlayer);
    ClearAllFactionMembers(oKiller, oPlayer);
    DelayCommand(2.5, PopUpGUIPanel(oPlayer, GUI_PANEL_PLAYER_DEATH));
}

```

Notice I changed the amount in the ClearAllFactionMembers() function from 100 to 80. setting it at 100 will make that faction 100% (green) to the PC and if the game is set on Non-PVP settings, the PC will not be allowed to attack that faction again. Also I added 2 lines just before the last line - DelayCommand. The First line gets the last object taking hostile actions towards the PC and assigns it to the variable oKiller, the second line calls the ClearAllFactionMembers() function and it uses the custom faction of the NPC oKiller, and adjust it back to neutral towards the PC. This function too will set the NW_G_Playerhasbeenbad variable for your custom factions if you choose to use it. Note that care must be taken if using this function because it will set any faction to "like" the PC, including hostile factions that should remained hostile. This can be handled by

checking the faction of the killer first and skip the Clear.

Now, oKiller will be the last creature that killed the PC and not the module. Also, when the PC dies the faction will clear just as it does for the standard factions. Note that I used `GetLastHostileActor(oPlayer)`; instead of `GetLastKiller()`; This is because `GetLastKiller()` only returns a valid object to the *caller* of the script. Since the `OnPlayerDeath` script is attached to the module, the module is the caller. Nothing killed the module therefore `GetLastKiller()` will be invalid. However, `GetLastKiller()` can be used if we make the player that died call the `OnDeath` script via the `ExecuteScript()` function, but that's for more advanced scripting

Now I will cover the `OnPlayerRespawn` script and cover what happens when the Player hits the "Respawn" button. Again there is a lot of "left Over" stuff from the OC. Here I will post a Respawn script with a lot of the junk stripped out.

```
#include "nw_i0_plot"
// * Applies an XP and GP penalty
// * to the player respawning
void ApplyPenalty(object oDead)
{
    int nXP = GetXP(oDead);
    int nPenalty = 50 * GetHitDice(oDead);
    int nHD = GetHitDice(oDead);
    // * You can not lose a level with this respawning
    int nMin = ((nHD * (nHD - 1)) / 2) * 1000;
    int nNewXP = nXP - nPenalty;
    if (nNewXP < nMin)
        nNewXP = nMin;
    SetXP(oDead, nNewXP);
    int nGoldToTake = FloatToInt(0.10 * GetGold(oDead));
    // * a cap of 10 000gp taken from you
    if (nGoldToTake > 10000)
    {
        nGoldToTake = 10000;
    }
    AssignCommand(oDead, TakeGoldFromCreature(nGoldToTake, oDead, TRUE));
    DelayCommand(4.0, FloatingTextStrRefOnCreature(58299, oDead, FALSE));
    DelayCommand(4.8, FloatingTextStrRefOnCreature(58300, oDead, FALSE));
}
void main()
{
    object oRespawner = GetLastRespawnButtonPresser();
    ApplyEffectToObject(DURATION_TYPE_INSTANT, EffectResurrection(), oRespawner);

    ApplyEffectToObject(DURATION_TYPE_INSTANT, EffectHeal(GetMaxHitPoints(oRespawner)),
    oRespawner);
    RemoveEffects(oRespawner);
    /* Return PC to temple
    string sDestTag = "NW_DEATH_TEMPLE";
    if (GetIsObjectValid(GetObjectByTag(sDestTag)))
    {
        if (sDestTag == "NW_DEATH_TEMPLE")
        {
            object oPriest = GetObjectByTag("NW_DEATH_CLERIC");
            //SetLocalInt(oPriest, "NW_L_SAYONELINER", 10);
            //AssignCommand(oPriest,
            DelayCommand(3.0, ActionStartConversation(oRespawner));
            AssignCommand(oPriest, DelayCommand(2.1,
            PlayVoiceChat(VOICE_CHAT_TALKTOME,
            oPriest));
            SetLocalLocation(oRespawner, "NW_L_I_DIED_HERE",
            GetLocation(GetLastRespawnButtonPresser()));
            SetLocalInt(oRespawner, "NW_L_I_DIED", 1);
```

```

        SetLocalObject(oPriest, "NW_L_LASTDIED",
GetLastRespawnButtonPresser());
        // * April 2002: Moved penalty here, only when going back to the death
temple
        ApplyPenalty(oRespawner);
    }
    object oSpawnPoint = GetObjectByTag(sDestTag);
    AssignCommand(oRespawner, JumpToLocation(GetLocation(oSpawnPoint)));
    // * mak
    }
    // else do nothing, just 'res where you are.
}

```

First we see a include file "#include "nw_i0_plot". This file holds the function RemoveEffects() which removes all negative effects on the Player, much like the Raise() function in the OnPlayerDeath script. Then we see a function called ApplyPenalty(). This function is used if the NW_DEATH_TEMPLE object is placed in your module, more on that in a bit. Next we see that the player is raised and healed and all bad effects are removed.

In the OC game, when you respawned, the PC would transport to the temple. This can be done in your own module as well simply by placing a waypoint, or any object for that matter with a tag of "NW_DEATH_TEMPLE". If this object is valid, then the player will be teleported to it on respawn AND given a respawn penalty by the ApplyPenalty() function. If you want the player to port here, but no penalty, then simply comment out the line ApplyPenalty(). If this object is not valid, nothing happens and the PC respawns where they died. The object oPriest and other variables within that if block can be used if wanted but a part of the OC and can be taken out as well. If you want to Apply the penalty without using the NW_DEATH_TEMPLE then add *else ApplyPenalty(oRespawner);* between the last 2 "}" at the end of the script.

That's the basics of how the bioware Death and respawn system works. By altering these 3 scripts, you can do just about anything for players dying, death, and re-spawn. Hopefully this information was helpful, enjoy

What does the ? mean in NW Script?

The question mark (?) is called the **ternary conditional** and is a form of selection, much like an **if/else** assignment. The Lexicon has a brief but informative [article](#) covering the basic syntax.

You are probably familiar with using **an if/else** statement to check some condition and assign one of two values to a variable depending on the result: the ternary conditional does the same thing using just one line! Therefore if used wisely the ternary conditional can simplify or clarify some scripts. However it is one of the less well-known constructs exercise caution and comment well.

For example, if you wanted check if a number was odd or even you might write the following script:

```

string sResult;

// determine if number is odd or even using modulus
if(nNumber % 2)
{
    sResult = "Odd";
}
else
{
    sResult = "Even";
}

```

and while there is nothing wrong with that you might find it a little less cumbersome to rewrite it using the ternary conditional:

```

// determine if number is odd or even using modulus
string sResult = (nNumber % 2) ? "Odd" : "Even";

```

You can also nest ternary conditionals: much in the same way as you can with an **if/else** statement. However I would generally advise against this as it tends to result in greater confusion rather than greater clarity.

For example, if you wanted check if a number was positive, negative or zero you might write the following script:

```

string sResult;

// determine if number is odd or even using modulo
if(nNumber > 0)
{
    sResult = "Positive";
}
else if(nNumber < 0)
{
    sResult = "Negative";
}
else
{
    sResult = " Zero ";
}

```

which again could be rewritten:

```

// determine if number is positive, negative or zero
string sResult = (nNumber > 0) ? "Positive" :
    (nNumber < 0) ? "Negative" : "Zero";

```

VII. Appendix II: Code Error Explanation

By Tarne

Many people are joining BioWare's Neverwinter Night's Community daily, so it's no surprise more and more people are in need of C++ help.

Well, having been through this experience, I decided I should try and make this better for people in general with a simple list of 'How the hell do I work it' rules for the technically illiterate.

Let's begin with a list of errors and possible fixes. This is not a COMPLETE list, however I'll try to keep the most common ones up here.

Lets start out with the most obvious one:

UNKNOWN STATE IN COMPILER.

This means, that the engine has no clue of what you're trying to do. The areas of screw-ups you could do to cause this are huge, however there are a few awful common ones:

ONE: You're missing a closing bracket. If you open a statement, you need to close it at the end. Let me show you:

```
void main()
{ //This is opening a statement
object oSelf = OBJECT_SELF(); //Declaring oSelf in this single
script to mean 'OBJECT_SELF'.
{ //Opening another statement, for no real reason.
AssignCommand(oSelf, SpeakString("Hi Mom!", TALKVOLUME_SHOUT));
} //Closing the First statement
} //Closing the Second statement
```

Now, a few people may read that and go "Wait, what DOES that do?". Well, lets say you put that in a Heartbeat command on an NPC, like, say, a Badger. Ever six seconds, the badger is going to say 'Hi Mom'. Anyway, that's a quick clarification on that.

Next:

Closing a statement in a command or anything of the nature.

Every time you open with a '(' , there needs to be a ')' somewhere in there in order for it to compile.

Basically, it's like opening and closing the front door of your house. You open it to let fresh air in or go outside, and you close it because you realize the neighbor's dog will be

maiming your couch in a short period of time if you don't.

A quick example:

```
void main()//Open/Close! Even if you're not sure why!  
{//Opening Statement  
AssignCommand(GetObjectByTag("Something"), SpeakString("Hello,  
I'm 'Something', but nothing in particular.", TALKVOLUME_TALK));  
}//Closing Statement
```

Now, you're confused again. Well, let me unconfuse you real-fast:

When you ask it to perform a function, such as 'GetObjectByTag' and retrieve information, you need to open the function. These aren't the terms it designed in, but who cares, as long as you understand, right? Good, shut-up and learn.

So basically: GetObjectByTag ("Something") would mean that you are asking it to find the object tagged Something. Now when you enter the specifics of the command in for it, you need to have the '(' opening and ')' closing of that instance.

Of course, You don't need to close EVERYTHING after EVERY instance. See how it closes three times at the very end of the 'command'? Well, that's closing every door I opened. I leave some doors open that're in the active function, but when you ask it to retrieve information, you need to close the door after you tell it where to find the info.

It's like a Jehova's Witness asking for directions. You want that door closed ASAP when you're done, if you even bothered to open it.

Next:

Bad scripting in general.

Now, here's an example of something that can't be fixed but may be tried by the extremely computer illiterate:

```
make monster walk to bed
```

This may evolve to:

```
void main()  
{  
}  
make monster wet bed
```

Regardless, this isn't scripting, the computer has no clue what you're doing and it probably wishes you would stop. Alright, those are the most common "UNKNOWN STATE IN COMPILER". Lets move on to the next one:

ERROR: PARSING VARIABLE LIST

Now, first, what it means: When you see this, it means you're declaring the same variable in two instances or you're attempting to manipulate an already declared variable to something else.

Have no clue what I'm talking about?

Example:

```
void main()
{
object oBilly = GetLastUsedBy();
object oSussy = GetLastUsedBy();
}
// OR IT COULD BE...
void main()
{
object oBilly = GetLastUsedBy();
object oBilly = GetObjectByTag("Billy");
}
```

Okay, this can't work. Billy and Sussy aren't the same people, they aren't even the same gender, however, they are related, they're both declared variables. The computer recognizes this as a parsing variable because you're trying to declare it twice, so it would get confused when one or the other was called on different instances, when you don't even need it to do so.

SO, how do we fix this? Well, this one is easy, it involves the delete key or a simple button press. You can't declare two of the same type of variable, nor can you write two variables to one declaration. SO:

```
void main()
{
object oBilly = GetLastUsedBy();
}
```

Fixed the first one...

```
void main()
{
object oBilly = GetLastUsedBy();
object oBilly1 = GetObjectByTag("Billy");
}
```

Now, by deleting the extra instance or by changing the declaration name even slightly, we fix the problem. Simple, isn't it?

Next:

ERROR: NO LEFT BRACKET

You may have one to many or one to few doors open. Double check under the rules set

above.

ERROR: NO RIGHT BRACKET

You left the door open again.

ERROR: NO SEMI-COLON AFTER EXPRESSION

A Semi-Colon, a ';' for the illiterate is used to null an ACTIVE expression.

"What the hell does that mean?"

Well, imagine you've got a baseball bat, and you're pissed off at your neighbor, so you wail on your neighbor's car with ActionWailNeighborsCar. Well, each blow requires it's own action, so, you have to lift the bat back up off the car, but if you don't null statement, you'll sit there with your bat jammed in the car till the cops get there.

Example:

```
void main()  
{  
ActionOpenDoor(GetObjectByTag("CarDoor"));  
}
```

"What does this one do?"

Well, it's an example: You see, once the door is open, we need to stop opening it, close it, then drive away before the cops show up. So essentially, whenever you perform an action, you'll want to stop performing it when it's done with a null statement.

ERROR: NO NULL STATEMENT AFTER "IF"

Well, you asked a question, then suddenly, before they could answer, you told them not to, or covered your ears. The computer doesn't understand your logic, it wants to tell you the answer, because it likes to gossip.

"What DID I do wrong?"

```
void main()  
{  
if(GetLocalInt(GetFirstPC(), "question" == 0);  
}
```

Now, that WOULD work, in theory, had you not placed the ';' or NULL-STATEMENT at the end of the question. In other words, don't cover your ears after asking questions.

So the solution:

```
void main()
```



```
{
if(GetLocalInt(GetFirstPC(), "question" == 0)
}
```

And it works, just like that.

Next:

ERROR: TOO MANY "IF" STATEMENTS

Now, this is a pain, really. You want to set it up to make sure this doesn't happen 'if' and 'if' but 'if' and 'if', well, you get the idea, right? Right.

The problem:

```
void main()
{
if(GetLocalInt(GetFirstPC(), "question" == 0)
if(GetLocalInt(GetFirstPC(), "question2" == 0)
}
```

You're declaring two possible 'if' situations. You recognize this as being 'if' and 'if' because you have logic, however the computer, being illogically logical, is confused by it. It's simple to fix: You need to change it to include '&&' between the two.

Here's an example:

```
void main()
{
if(GetLocalInt(GetFirstPC(), "question" == 0
&&(GetLocalInt(GetFirstPC(), "question2" == 0))
}
```

Just how does that work? Well, we open a door, and some other doors, but at the end of the first instance of the 'if', we don't close the door. The same for all following possible, we leave them open and include '&&', the double statement for making damn sure the computer knows to include the second instance.

Next:

ERROR: UNIDENTIFIED INDENTIFER

Isn't that an oxymoron? Well, it's trying to tell you that you're using a command in the statement that makes no sense whatsoever to it. It may be caused by a simple typo.

Example:

```
void main()
{
object oPC = GetPCSpeaker();
```

```
ApplyEffectToObject(DURATION_TYPE_INSTANT, EffectKnockDown, oPC,
5.0);
}
```

Now you may ask, "What the heck is wrong with that? Why am I getting an error?" Well, you see, BioWare didn't include 'EffectKnockDown', they used 'EffectKnockdown', much the same as 'EffectBlind' won't work while 'EffectBlindness' will.

So basically, you just need to double check your identifier and compare it against what you know will work.

Sometimes, this isn't easy. You know what effects you can do, but what about visual effects? You may go insane trying to guess them, but there's an actual list of them too! It's in your 'Source' file in the visual.2da file.

Next:

ERROR: SKIPPING DECLARATIONS VIA "case" DISALLOWED.

Now, this is a frustrating one to guess. It doesn't actually tell you what's wrong. Whenever you're using a 'CASE' set-up, and you want to have a pre-defined set of declarations, they need to come before the 'CASE'.

"What the hell are you talking about?"

Example:

```
void main()
{
int nVariable
{
switch(nVariable)
{
case 1:
if(GetLocalInt(GetFirstPC(),"Variable" == 0)
object oPC = GetFirstPC();
ApplyEffectToObject(DURATION_TYPE_INSTANT,EffectKnockdown,oPC,5.0
);
break;
}
case 2:
{
if(GetLocalInt(GetFirstPC(),"Variable" == 0)
ApplyEffectToObject(DURATION_TYPE_INSTANT,EffectBlindness,oPC,5.0
);
break;
}
}
}
```

Now, you may be confused, why this wouldn't work. Well, there are probably a couple of reasons, but for the first and most prominent one, the error you're getting, is the fact you're making a declaration in the case itself. That's not allowed, so to fix it...

Example:

```
void main()
{
    object oPC = GetFirstPC();
    int nVariable
    {
        switch(nVariable)
        {
            case 1:
                if(GetLocalInt(GetFirstPC(),"Variable" == 0)

ApplyEffectToObject(DURATION_TYPE_INSTANT,EffectKnockdown,oPC,5.0
);
                break;
            }
            case 2:
                {
                    if(GetLocalInt(GetFirstPC(),"Variable" == 0)

ApplyEffectToObject(DURATION_TYPE_INSTANT,EffectBlindness,oPC,5.0
);
                    break;
                }
            }
        }
    }
```

So, how did I fix it? Well, I simply moved the declaration to fit inside of the first case. So now, it works, in principle.

Next:

ERROR: MUST INCLUDE "void main" (or something like that)

Well, if you wrote a script to be included in the main, you probably figured out what the problem was, but then, maybe you didn't and you're reading this.

For some unknown reason, your compiler would like everything to include 'void main'. I don't know why, I don't really care, but I'll tell you how to fix the error.

Lets say you've got a neat little script included in the main, I'll use one from Visage of Nightmares to demonstrate:

```
void FireDiscipline(object oPC)
{
    effect eFire1 = EffectAbilityIncrease(ABILITY_STRENGTH, 5);
    effect eFire2 = EffectAttackIncrease(10);
    effect eFire3 = EffectDamageIncrease(5,
```

```

DAMAGE_TYPE_SLASHING);
    ApplyEffectToObject(DURATION_TYPE_TEMPORARY, eFire1, oPC,
10.0f);
    ApplyEffectToObject(DURATION_TYPE_TEMPORARY, eFire2, oPC,
10.0f);
    ApplyEffectToObject(DURATION_TYPE_TEMPORARY, eFire3, oPC,
10.0f);
    AssignCommand(oPC, DelayCommand(10.0f, FireDiscipline(oPC)));
} //Fire Discipline

```

Now, you don't want this to be outside of the main, otherwise it won't work the correct way. But you're getting a compiling error, because you need to include the 'void' for it to compile. So how do you fix it? Just watch:

```

void FireDiscipline(object oPC)
{
    effect eFire1 = EffectAbilityIncrease(ABILITY_STRENGTH, 5);
    effect eFire2 = EffectAttackIncrease(10);
    effect eFire3 = EffectDamageIncrease(5,
DAMAGE_TYPE_SLASHING);
    ApplyEffectToObject(DURATION_TYPE_TEMPORARY, eFire1, oPC,
10.0f);
    ApplyEffectToObject(DURATION_TYPE_TEMPORARY, eFire2, oPC,
10.0f);
    ApplyEffectToObject(DURATION_TYPE_TEMPORARY, eFire3, oPC,
10.0f);
    AssignCommand(oPC, DelayCommand(10.0f, FireDiscipline(oPC)));
} //Fire Discipline
void main()
{
}

```

Now you might be sitting here with a dumbfounded look on your face, or possibly just glaring at the computer for the illogical logic of that, but, it still works.

Now, there are other instances, when at times, you may have an error, try to add the fix to the line with the error and it doesn't work, infact, it makes things worse.

Well, the computer isn't always the brightest... In some instances, it'll mark errors for times when you forget 'Null-Statements' or ';' on the next line over. So if you get that error, assume it's on the line just above the one marked as 'flawed'. Technically, that IS the location of the error, however, the way to fix it is just above that line.

These are the most common of the errors that come to mind, and if you still can't compile after reading these explanations, then feel free to ask the board right away, however, try and use these fixes first.

Hope this helps a few people.

VIII. Appendix III: Waypoints Explained

By baron of gateford

WAYPOINTS USED WITH WALKWAYPOINTS (or normal waypoints)

There are 4 different types of waypoint to get your NPC to move around in a set pattern via the standard Bioware Waypoint system.

WP_ , WN_ , POST_ , NIGHT_

Each waypoint has a prefix (as shown above) followed by the TAG of the NPC.

So if your NPC had a tag of SMITH then the waypoint tags would be,

WP_SMITH, WN_SMITH, POST_SMITH, NIGHT_SMITH

WP_ Waypoints: These waypoints are used to move your NPC around in the day.

WN_ Waypoints: These waypoints are used to move your NPC around at night.

POST_ Waypoints: Used to make your NPC walk to one spot and stay there at daytime.

NIGHT_ Waypoints: Used to make your NPC walk to one spot and stay there at nighttime.

The NPC can have multiples of the WP_ and WN_ waypoints, but only one POST_ and one NIGHT_ waypoint.

Multiples of the WP and WN waypoints have a sequence number at the end of the tag name, so for the NPC called SMITH, the waypoints would be;

WP_SMITH_01, WP_SMITH_02, WP_SMITH_03, etc....

The POST waypoint cannot be used with WP waypoints for a given NPC.

The WP waypoints override POST waypoints. A similar rule applies to NIGHT

waypoint not being used with WN waypoint WN waypoints override NIGHT waypoint.

This means you cannot have the NPC run a set pattern of WP_ waypoints to then stand at a POST_ waypoint.

The easiest way of getting the waypoints up and running for your npc is;

Right click on your npc and choose create waypoint.

A waypoint is created underneath or near the npc.

Move the mouse to the point on screen you want your second waypoint to appear, and then right click, the option to create a waypoint is given. Choosing create waypoint will create a second waypoint for your NPC. This process can be done multiple times.

To get a guard to return to his barracks at night.

If the guard has a tag of GUARD, then place a NIGHT_GUARD waypoint in the barracks. This can be used with a sequence of WP waypoints for your guard to patrol in the day, e.g. WP_GUARD_01, WP_GUARD_02, etc

Or it can be used with a POST waypoint tagged POST_GUARD to have him stand at his post in the day.

To allow for cross area transitions, e.g. Barracks in one area, patrol route in another.

You will need to set the global int X2_SWITCH_CROSSAREA_WALKWAYPOINTS to one on the module.

To do this:

In module properties, click the advanced tab, then click variables.

Enter X2_SWITCH_CROSSAREA_WALKWAYPOINTS as an int and the value to 1.

My guard gets stuck crossing area transitions.(or is going a strange route)

You will have to experiment with this, due to some issues with the pathfinding.

If your area has more than one transition point to a second (but same) area then do not expect your NPC to use the transition point you want.

It may be best to experiment using POST and NIGHT waypoints first to see which route the NPC favors. Its worth knowing this, so you know why its happening, and you don't waste too much time, like I have

My NPC is getting stuck on walls.

This is due to the AI on NPC's being set to low if a player is not in the area.

One of the best ways to get around this is to have more WP, WN waypoints if you can to help the NPC pathfind a lot better. Obviously this won't help POST and NIGHT waypoints, but if you know scripting you could place a script on enter of an area to check if any players are in the area and if not, jump the NPC to its POST or NIGHT waypoint.

My NPC is not facing the same direction as his waypoint when at a POST or NIGHT waypoint.

Right-click the Waypoint and go to variables.

Enter X2_L_WAYPOINT_SETFACING as an int and the value to 1.

My NPC is not changing to its NIGHT_ or WN waypoints at night time.

Ensure that in the onspawn script of your NPC, the following line is uncommented (e.g. the // removed before it, and the line not displayed in green)

```
SetSpawnInCondition (NW_FLAG_DAY_NIGHT_POSTING);
```

My NPC doesn't seem to want to walk to their waypoints

In the onspawn script of your NPC ensure that following two lines are still commented out (e.g. the // is still before the lines, and the line is displayed green)
SetSpawnInCondition (NW_FLAG_IMMOBILE_AMBIENT_ANIMATIONS);
SetSpawnInCondition (NW_FLAG_AMBIENT_ANIMATIONS);

WAYPOINTS USED WITH MOBILE AMBIENT ANIMATIONS

The following Waypoints require that mobile ambient animations be switched on at spawn time of the NPC.

That means that the following line needs to be uncommented in the onspawn script. (e.g. the // removed before it, and the line not displayed in green)

SetSpawnInCondition (NW_FLAG_AMBIENT_ANIMATIONS);

When mobile ambient animations is switched on for a NPC, the NPC will wander around and interact with other NPC's that have either mobile or immobile ambient animations switched on. They will also interact with nearby placeables. walking up to a placeable, closing nearby doors.

Also worth noting that the NPC's will also use the same animations for mobile ambient animations, as they do for immobile ambient animations (with a bit more moving around).

The NPC with mobile ambient animations switched on will never wander too far from its spawn point(starting position), always returning back to it.

The following waypoints allow for better control of your NPC's. To wander around more between the waypoints described below, as well as the placeables the NPC comes across, or the other NPC's, or its spawn point(start position).

Remember - The NPC's will not walk straight to these waypoints as in walkwaypoints above but will meander amongst them(wander about them in a casual laid back way).

Generic Stop Waypoint

This waypoint is tagged "NW_STOP".

If you place a few of these in an area, lets say a city exterior for example.

If the NPC's spawned in that area have mobile ambient animations switched on, they will walk amongst the NW_STOP waypoints. This works quite well if you want a busy/crowded city/town feel.

So you could have a park area in the city with an NW_STOP waypoint, or/and a NW_STOP waypoint near a market stall.

The NPC's spawned in that are will at some point in their wanderings go to these waypoints.

Tavern Waypoint

This waypoint is tagged "NW_TAVERN".

If NPC's are spawned inside an area with the tavern waypoint, then those NPC's will remain in the tavern and not leave it.

However, for NPC's spawned outside the tavern area, If you place a NW_STOP waypoint inside the tavern as well (as the NW_TAVERN), The NPC's will cross the area transition to visit the NW_STOP waypoint in the Tavern, neatly closing the door behind them on entry/exit of the tavern/area outside.

There has been a bit of a misconception regarding this waypoint, this waypoint does not make any new animations appear as far as I know by simply dropping it in a tavern. Only by setting the immobile or mobile animations on an NPC in the spawn script will the NPC move.

Shop Waypoint

This waypoint is tagged "NW_SHOP".

Like the tavern waypoint, if an NPC is spawned inside an area with the shop waypoint then the NPC will remain in the shop and not leave it (the shop keeper/assistant I guess).

If an NPC is spawned outside the shop area, If you place a NW_STOP waypoint inside the shop as well, the NPC will cross the area transition to visit the NW_STOP waypoint, neatly closing the door behind them on entry/exit of the shop.

Home waypoint

This waypoint is tagged "NW_HOME".

If an npc is spawned in an area that that has a home waypoint then the NPC will class that area as his home. What should happen is the NPC will leave the area in the day and return to it at night. After long testing, the NPC only seemed to want to leave the area if a player entered it, however the NPC did return back to the area at night.

Safe Waypoint

This waypoint is tagged "NW_SAFE".

Placing one of these waypoints in area, means that any injured NPC with mobile animations will retreat to this point. Sort of like a cowering corner. NPC's will not cross area transitions to get to a NW_SAFE waypoint. If there is not a NW_SAFE waypoint in the same area then they will resume mobile animations.

Stealth Waypoint

This Waypoint is tagged "NW_STEALTH".

Placing one of these waypoints in an area, means any NPC with stealth capabilities that reaches one of these waypoints, turns stealth mode on. Could be good for using in dark caverns/dungeons etc, to have NPC disappear into and appear from the darkness.

Detect waypoint

Ok, I admit, this one I thought I'd leave you guys to figure out.....

My NPC is not wandering around even though mobile animations is switched on.

Ensure that the following line is commented out in the onspawn script of the NPC.
(e.g. the // is still before the lines, and the line is displayed green)
SetSpawnInCondition (NW_FLAG_IMMOBILE_AMBIENT_ANIMATIONS);

Can I use NW_STOP waypoints inside NW_TAVERN, NW_SHOP waypointed areas?

The answer is yes.

You will need to place NW_STOP waypoints in the NW_TAVERN, NW_SHOP areas if you want the NPC's outside to come and drink/shop

Creature related waypoints

As an extra, I thought I would throw this one in, I've done most of the others, so.....
The following waypoint is used with the Bioware Beholder AI.

Beholder AI exit waypoint

This waypoint is tagged "X2_WP_BEHOLDER_TUNNEL"
If you look in the script "x2_ai_behold" it tells you what the waypoint is for. It explains;
Beholder will always use its eyes, unless
a) If threatened in melee, it will try to move away or float away
to the nearest waypoint tagged X2_WP_BEHOLDER_TUNNEL

So if you place a few of these waypoints around, your beholder will escape to the nearest one. If you noticed in Chapter two of HotU these waypoints where placed on the other side of room doors. (sneaky)

VIII. Appendix III: Important Resources

NWN Lexicon

<http://www.nwnlexicon.com/>

It really doesn't get much more important than this. This details just about everything there is to know about NWScript. Important to know when a script won't work and it should and you find out the function doesn't... er function.

NWVault Scripting University

<http://nvwault.ign.com/?dir=resources/scripting>

Very nice tool to learn how to script.

#nwscript

irc.neverwinterconnections.com

A chat room where you can get help with scripting problems.