

Neshke Narovken's
Extendable Spawn System

Tutorial

by Mark "Black Cloud" Morris

version 1.1
April 1, 2004

History

Version 0.50 - Tutorial for ANESS v. 1.2.2 published December 2, 2003. Several incomplete sections.

Version 1.00 - Tutorial for NESS 8.0 and published on February 1, 2004. Completed sections not finished. Made updates for new version 8.0 of NESS.

Version 1.10 - Tutorial for NESS 8.0 and published on March 31, 2004. Corrected numerous grammar and spelling errors. Corrected a no longer existing bug regarding the use of PC flag. Expanded and clarified definitions for AL, FC, PT, HL flags. Has **not** been upgraded to include NESS 8.1 yet.

Copyright (c) 2004 Mark Morris.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts.

A copy of the license is included in the section entitled "Appendix A - GNU Free Documentation License."

Table of Contents

TABLE OF CONTENTS	III
TABLE OF FIGURES	VI
AUTHOR'S NOTE	VIII
CHAPTER 1 - INTRODUCTION	1
WHAT IS NESS	1
CHAPTER 2 – CREATING A LEARNING MODULE AND INSTALLATION	2
CREATING A DEMO MODULE	2
WHERE TO GET NESS	3
INSTALLING NESS	3
WHAT YOU GET	4
OTHER USEFUL APPLICATIONS.....	4
CHAPTER 3 - CHOOSING YOUR SPAWN	5
SPAWNING STANDARD CREATURES.....	5
SPAWNING CUSTOM CREATURES	6
CHAPTER 4 - SPAWNING BASICS - SP, SN, AND SA	8
SP[N] OM – SPAWN POINT	8
SNN MM – SPAWN A SPECIFIC NUMBER.....	9
SA[N] MM – SPAWN AT ONCE.....	10
CHAPTER 5 - SPAWNING DAY AND NIGHT – DY, DO, NO, AND HR	12
DYN TM – SPAWN ON A SPECIFIC DAY OF THE MONTH.....	12
HRN TM – SPAWN AT A SPECIFIC HOUR	12
DO D – SPAWN ONLY DURING THE DAY	12
NO D – SPAWN ONLY DURING THE NIGHT	13
CHAPTER 6 - PATROLLING – PR	15
PRN TN C – SPAWN A CREATURE TO PATROL A ROUTE	15
TURNING OFF PR.....	18
EXTRA CREDIT – MAKING THE PATROLLER DO SOMETHING.....	18
CHAPTER 7 - WALKING AND DELAYING – RW, SD, AND IS	24
RW RN – RANDOM WALK	24
TURNING OFF RW	24
RH DN – RETURN HOME AFTER ACTION	24
SDN MM P – DELAY SPAWNING.....	25
ISN DM – INITIAL SPAWNING STATE	26
CHAPTER 8 - SPAWN PLACEMENT – SR, SL, SF, ST, AND RS	27
SRN MM P – SPAWN WITHIN A RADIUS.....	27
SLN RM – SPAWN AT ANOTHER LOCATION	27
SF – SPAWN FACING.....	28
ST – SPAWN SITTING.....	28
RSN – RANDOM SPAWN.....	29
CHAPTER 9 - ENTRANCE AND EXITS – EE AND EX	30

EEN RM – ENTER AND EXIT AT WAYPOINT EEN	30
EXN RM – EXIT AT WAYPOINT EXN	30
CHAPTER 10 - PREPARED FOR DEATH – CD.....	31
CDN TM RQ D – CORPSE DECAY	31
CHAPTER 11 - SPAWN UNSEEN – PC, SU, AND TR.....	34
PC[N] R – SPAWN ONLY IF A PC IS IN THE AREA	34
SUN I M – SPAWN ONLY IF PCs ARE MORE THAN N METERS AWAY	37
TRN DM – PC PROXIMITY TRIGGER.....	37
CHAPTER 12 - SPAWN SPECIAL CHILDREN – AL, FC, PT, CL, AND HL.....	40
ALN SM – SPAWN CREATURES WITH A SPECIFIC ALIGNMENT	40
FCN – SPAWN CREATURES WITH A SPECIFIC FACTIONS.....	41
PT – SPAWN CHILDREN AS PLOT OBJECTS.....	41
CLN MM – SPAWN CHILDREN WITH A SPECIFIC LIFE SPAN	41
HL[N] E – HEAL CHILDREN	42
CHAPTER 13 - SPAWNING THE RICH – RG.....	43
RGN MM CQ – SPAWN WITH GOLD	43
CHAPTER 14 - SPAWNING PLACEABLES AND ITEMS – PL AND IT.....	44
PL[N] TM PQ – SPAWNING PLACEABLES	44
IT – SPAWNING ITEMS.....	45
CHAPTER 15 - DEACTIVATING THE SPAWNED – DS AND DI.....	46
DSN SM_DIQ – DEACTIVATE A SPAWN.....	46
CHAPTER 16 - SPAWNING WITH EFFECT – AE, FX, AND OE.....	47
AEN DM – SPAWN WITH AN AREA EFFECT	47
FXN DM – SPAWN IN WITH EFFECT	49
OEN DM – SPAWN WITH OBJECT EFFECT	50
CHAPTER 17 - SPAWNING A MERCHANT – SM.....	52
SM – SPAWN A MERCHANT	52
CHAPTER 18 - LOOT TABLES – LT.....	55
LTN AM BQ CR – SPAWN LOOT ON CREATURES USING LOOT TABLES.....	55
USER DEFINED LOOT TABLE SCRIPTS.....	55
DANIELEB'S MERCHANT BASED LOOT TABLES.....	55
CYRUS_1469 DMG LOOT TABLES.....	57
CHAPTER 19 - SPAWNING A GROUP – SG.....	61
SG – SPAWN A GROUP	61
DIFFERENT LOOT FOR DIFFERENT CREATURES IN A GROUP	64
SPECIFIC CREATURES WITH SPECIFIC LOOT IN A GROUP	64
CHAPTER 20 - SPAWNING A CAMP – CM.....	66
CM DN – SPAWN A CAMP WITH CREATURES AND PLACEABLES.....	66
CHAPTER 21 - FLAG TABLES – FT.....	71
FT[N] – USE FLAGS FROM FLAG TABLE	71
CHAPTER 22 - SPAWN SCRIPT – SS.....	74
SSN DM – RUN AN ONSPAWN SCRIPT WHEN CHILDREN ARE SPAWNED OR DESPAWNED.....	74

SPAWN INVISIBLE	74
CHAPTER 23 - HEARTBEAT SCRIPT – HB.....	75
HBN – CREATURES WILL RUN A SCRIPT ON HEARTBEAT	75
CHAPTER 24 - CUSTOM CODE – CP AND CC.....	76
CPN RM – CHECK PC CUSTOM SCRIPT TO DETERMINE IF SPAWN PROCEEDS.....	76
CCN – CHECK CUSTOM SCRIPT TO DETERMINE IF SPAWN PROCEEDS.....	76
CHAPTER 25 - DEATH SCRIPTS – DT.....	78
DTN – RUN SCRIPT ON DEATH OF CREATURE	78
CHAPTER 26 - PROPER ID AND FLAGGING THE FLAG – ID AND CF.....	79
IDN – GIVE A SPAWN WAYPOINT AN ID NUMBER.....	79
CF – STORE FLAG INFORMATION.....	80
CHAPTER 27 – FUNCTIONS FOR EXTERNAL CONTROL.....	82
CHAPTER 28 - SPAWN BANNERS.....	85
INSTALLING SPAWN BANNERS.....	85
USING SPAWN BANNERS.....	85
APPENDIX A – GNU FREE DOCUMENTATION LICENSE.....	88
APPENDIX B – APPROPRIATE PARAMETER RANGES FOR FLAGS	93
APPENDIX C – SPAWN_README FILE.....	95
APPENDIX D – CONTENTS OF NESS8_0FULLERF.....	111
APPENDIX E – VISUAL EFFECT CONSTANTS.....	113
APPENDIX F – AREA EFFECT CONSTANTS	118
APPENDIX G – CYRUS' & DANIELEB'S LOOT TABLES	119

Table of Figures

Figure 1 – Plan of <i>City</i>	2
Figure 2 – spawn_sample_hb script.....	3
Figure 3 – Script to be included in OnHeartBeat event script.	3
Figure 4 – Build Module dialog box.	4
Figure 5 – Waypoint properties.....	5
Figure 6 – Our first example.	6
Figure 7 – Irene, the dire rat, and the fire beetle.	6
Figure 8 – Custom dwarf spawn waypoint.....	6
Figure 9 – Custom dwarf properties dialog box.....	7
Figure 10 – Dwarf at work.	7
Figure 11 – Three dire rats.	9
Figure 12 – Three dire rats, minimum one.....	10
Figure 13 - Twelve dire rats, three at a time.	10
Figure 14 – Irene and the twelve dire rats.....	11
Figure 15 – Forest area.....	12
Figure 16 – Three skeletons night only.	13
Figure 17 – Three skeletons.	13
Figure 18 – Module properties.....	14
Figure 19 – Make some patrol waypoints.	15
Figure 20 – Dwarf patrol waypoint one.	15
Figure 21 – Dwarf on patrol.	16
Figure 22 – script_cc_patrol.....	18
Figure 23 – Check the door script.	19
Figure 24 – Custom dwarf scripts dialog box.	20
Figure 25 – OnUserDefine default NWN script.....	21
Figure 26 – Dwarf patroller conversation.	22
Figure 27 – OnUserDefine script for dwarf.	23
Figure 28 – Three skeletons walking random.	24
Figure 29 – Adjust Location dialog box.....	28
Figure 30 – Not so invisible chair.	28
Figure 31 – Invisible chair.	28
Figure 32 – Lootable beetle corpse.	32
Figure 33 – Item checked as droppable.....	33
Figure 34 – Four skeletons for PC reset test.	35
Figure 35 – Three dire rats with PC trigger.	37
Figure 36 – x out the dire rats.	38
Figure 37 – Fire beetle flags and placement.	38
Figure 38 – Placing a treasure chest.....	44
Figure 39 – Placing a magic sword item.....	45
Figure 40 – User defined section of spawn_cfg_fxae.....	47
Figure 41 – My user defined section of spawn_cfg_fxae.	48
Figure 42 – User defined section of spawn_cfg_fxsp.....	49
Figure 43 – My spawn_cfg_fxsp user defined section.....	50

Figure 44 – Actions Taken script to open merchant.	52
Figure 45 – Merchant Properties Basic.....	53
Figure 46 – Merchant Properties Advanced.....	53
Figure 47 – Merchant spawn waypoint.....	53
Figure 48 – User defined loot tables.	56
Figure 49 – Sample Cyrus loot table entry.....	57
Figure 50 – Custom Cyrus loot table entry.	58
Figure 51 – four_bad_guys group.	61
Figure 52 – User defined group four_bad_guys.....	62
Figure 53 – Scaled encounter group.....	63
Figure 54 – Variable loot.	64
Figure 55 – Specific creatures with specific loot.	65
Figure 56 – Skeleton camp waypoint.....	66
Figure 57 – User defined camps.....	69
Figure 58 – Random camp.	70
Figure 59 – User defined section of flag table.	72
Figure 60 – Trigger OnEnter event.	77
Figure 61 – Custom check script.....	77
Figure 62 – Five skeletons closing on PC.....	79
Figure 63 – spawn_cc_spawn run to spawn point script.....	79
Figure 64 – On Dialog Event script in the UserDefined script file.....	81
Figure 65 – Module properties dialog box.	85
Figure 66 – Create Rod of Spawn Banner.....	85
Figure 67 – Spawn banner conversation.	86
Figure 68 – Spawn banners.	86

Author's Note

The Extendable Spawn System uses a system of codes placed in the name field of a waypoint. These codes have been called options, parameters, and flags by various users. For clarity they will be called **flags** or **primary flags** in this tutorial. A list of the flags can be found in the `spawn_readme` file provided with the NESS system and listed in Appendix C. The term option will be used in conjunction with **optional flags** that are appended to the **primary flags**. These optional flags are listed under the primary flags in `spawn_readme` in Appendix C. Most flags have required or optional numeric parameters associated with them. These will be called **optional parameters** or **required parameters** as the case may be.

Brackets, [], surrounding a parameter indicate it is optional. Missing brackets indicate it is required.

SP[n] - indicates the SP flag has an optional integer parameter, n, that can follow it.
DYn - indicates the DY flag has a required integer parameter n.

Italics will be used to indicate text that is to be entered into a field or selected. This is to prevent any confusion that might occur regarding the use of quotes and if the quote should be entered as well in the field, etc.

A **spawn point** or **spawn waypoint** is the waypoint that defines a spawn and will begin with an SP in the name field. A **spawn location** is the actual location where a spawn occurs and is not necessarily the spawn point.

The spawn waypoint tag field will hold the resref of the creature, NPC, placeable, or item to be spawned. In the tutorial the figures will show this as the tag of the creature, NPC, placeable, or item in capital letters. Disregard this and use the resref that is always in lower case letters.

The ALFA NESS specific flags SX, SB, NL, and EL are not discussed as I am not familiar with them.

Thank you to:

Neshke for the system and many of the custom scripts,
Cereborn for continuing to update and add functionality to NESS,
Cyrus_1469 for the help including his loot table system,
Illysio for hosting the www.lrjonline.net/swg web site where NESS can be found along with other helpful applications,
Many others who's identity I don't know who contributed script.

Chapter 1 - Introduction

What is NESS

Neshke Narovken's Extendable Spawn System (NESS) is a set of scripts designed to simplify the task of creating creatures, placeables, and items in the BioWare computer game *Neverwinter Nights*. To use it, the game module creator must import the scripts into the game module and create waypoints with a special set of flags in the waypoint name field and the name of the creature or placeable in the waypoint tag field. Many useful creature and placeable spawns can be achieved with no scripting at all. The spawn system is very powerful, flexible, and extendable. With just a little additional scripting even more exciting spawns can be easily created.

Neshke ceased work on the spawn system about December 2002 with version 7.0. Beginning in December 2002, Cereborn of *A Land Far Away* (ALFA), fixed bugs and extended the NESS system from version 7.01 through 7.03 through January 2003. Beginning in February 2003 Cereborn began modifications and extensions to NESS for use in ALFA's persistent world game module. Cereborn published ALFA NESS version 1.0 (ANESS) in February 2003. Through September 2003 extension, additions, and bug fixes have led to version 1.2.2. On January 26, 2004 Cereborn updated ANESS 1.2.2 by adding functionality, fixing bugs, and updating some creature names. He decided to eliminate the separation of NESS 7.03 and ANESS 1.2.2 and combine them into one product called NESS 8.0. This tutorial is for NESS version 8.0.

The author assumes the following of the reader:

- you own and have installed a copy of *Neverwinter Nights* on your computer,
- you know how to use the BioWare Aurora *Neverwinter Nights* Toolset,
- you know how to play the game in *Dungeon Master* (DM) mode,
- you know how to create a module,
- you know how to create an area,
- you know how to add creatures, items, waypoints, terrain, etc., to an area,
- you know how to create conversations, and
- you know how to do basic scripting.

Chapter 2 – Creating a Learning Module and Installation

In this chapter we will show you:

- how to create a new module and one area for our spawning grounds,
- where to get NESS version 8.0, and
- how to import NESS into your module.

Creating a Demo Module

Let's create a module to use for our examples. Create a new module and name it *NESS Demo*. Create a new 8 x 8 tile city exterior area in the new module named *City*. Lay out the buildings and streets like those in Figure 1. Of course, you can always use your own area.



Figure 1 – Plan of *City*

Where to Get NESS

1. NESS 8.0 can be obtained on the internet at <http://www.lrjonline.net/swg/> a site hosted by fellow NESSer Illysis.
2. Select *NESS8_OFull.rar*. This will download the zipped file *NESS8_OFull.rar*.
3. Unzip the file and you will have *NESS8_OFull.erf*.

```
#include "spawn_main"
void main()
{
    Spawn();
}
```

Figure 2 – spawn_sample_hb script.

```
#include "spawn_main" //this line goes at the top of the
                        //your script with the other includes

Spawn();               //this line goes in the body of your
                        //area OnHeartBeat event script
```

Figure 3 – Script to be included in OnHeartBeat event script.

Installing NESS

1. Now you can import the erf file. Select *File > Import* from the menu bar. Locate the *NESS8_OFull.erf* file and import it. When the *Import Resources* dialog box appears, click OK. After the import has completed, look at the scripts listed on the left of the toolset and you will find 48 files; 47 beginning with the word *spawn*.
2. Open the properties dialog box for the *City* area we just created by right clicking on the name of the area. Select the *Events* tab. Insert the script named *spawn_sample_hb* into the *OnHeartBeat* event by clicking on the button with the three dots on the *OnHeartBeat* event line. Close the Area Properties dialog box.

The *spawn_sample_hb* script is shown in Figure 2. If you are importing the system into an existing area with an existing *OnHeartBeat* script, include the single function *Spawn()* into your existing script and include the file *spawn_main* in the include section.

3. From the main menu select *Build > Build Module*. The module will recompile all the scripts and you will get the dialog box shown in Figure 4. Save the module.

That's it. You're ready to start spawning.

What You Get

NESS 8.0 will import 29 scripts, and seven placeable objects for the spawn system. The placeable objects are invisible containers used as corpses. NESS will also import 23 scripts, one conversation, one item, and two placeables for the banner system. I will discuss the banner system in Chapter 28 - Spawn Banners. The scripts and resources imported are listed in Appendix D.

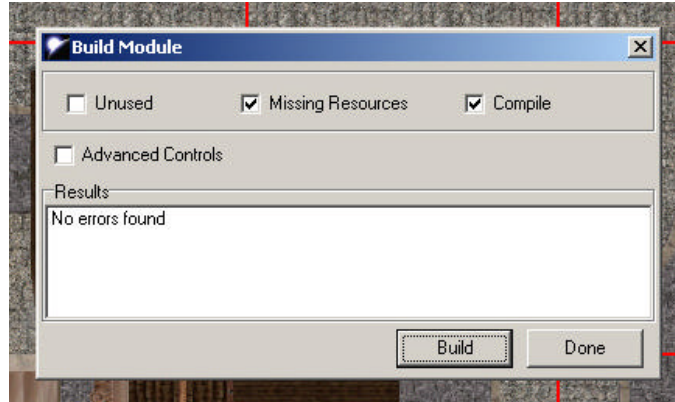


Figure 4 – Build Module dialog box.

Other Useful Applications

The **Spawn Waypoint Generator** is a Windows application for generating spawn waypoint flag sets for the name field of the waypoint. Along with the Spawn Waypoint Generator is a help file as useful as the generator application. Find both at <http://www.lrjonline.net/swg/>.

The **NWN Lexicon** help application is of great use to those who write script for NWN. Find it at <http://www.reapers.org/nwn/reference/>.

To summarize:

- you will need a practice module, so create one,
- get NESS 8.0 off the internet at <http://www.lrjonline.net/swg/>,
- install it by importing the *erf* file *NESS8_OFull.erf*, and
- rebuild your module before trying to play the game.

Chapter 3 - Choosing Your Spawn

In this chapter we will:

- learn how to create a basic spawn, and
- learn how to select the creatures that will be spawned.

OK, you have a new module named *NESS Demo* with one area named *City*. You have all the NESS 8.0 scripts imported into the module, and the *City* area *OnHeartBeat* event has the script *spawn_sample_hb* placed in it. You're ready to go.

Place a waypoint in the area at the end of the dead end street in the northwest corner. Open the properties dialog box for the waypoint.

All spawn points in NESS are waypoints. NESS uses two fields in the waypoint properties to control the spawn: the waypoint Name field and the waypoint Tag field.

The waypoint name is used for all the flags and options that control the way the spawn occurs and the creatures act. For NESS to recognize and use the waypoint, the name of the waypoint must always start with the flag *SP*. So put *SP* into the name field of the waypoint properties dialog box. If you do not start the waypoint name with *SP*, NESS will ignore it. Note that everything in the name field must be upper case.

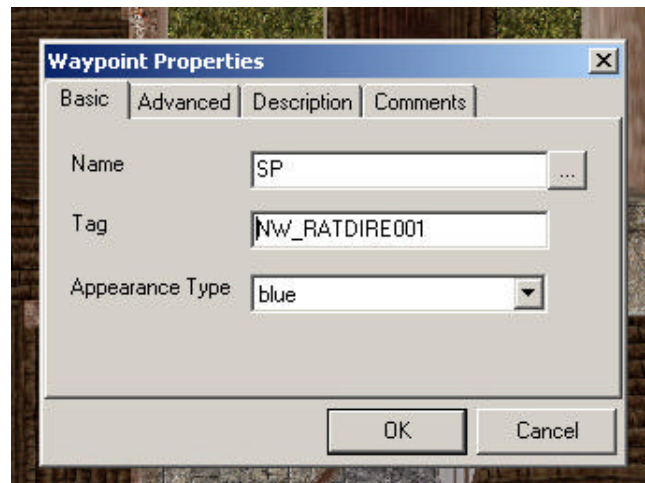


Figure 5 – Waypoint properties.

What kind of creature do we want to spawn? Let's make it something easy to kill, like a dire rat. Put the **resref** of the standard NWN dire rat in the tag of the waypoint. In this case *nw_ratdire001*. Your waypoint properties should look like Figure 5.

Close the waypoint dialog box. Note that the waypoint is now listed in the area contents box and is named *nw_ratdire001*. This makes it easy to see what will be spawned at each waypoint.

Place the PC starting point two tiles to the right, down the street from the waypoint and rotate it so that it faces the spawn waypoint, as shown in Figure 6.

Save the module. Start NWN in play mode, and open the *NESS Demo* mod. Make sure you select a character that can kill a dire rat. I recommend a ranger with a bow.



Figure 6 – Our first example.

Change the name field to *SP* and put the resref for a fire beetle, *nw_btlfire*, in the tag field. Close the waypoint dialog box and save the mod. Start NWN to play and open the demo module. Now we have a fire beetle and dire rat waiting for us. Kill them until you get bored and close the module.

Note that each fire beetle killed left behind a bag containing a fire beetle belly. This could create quite a mess after a short while. Later I will show you how to spawn beetles that will not leave anything behind.

The dire rat will appear on the first game heartbeat after the area loads. Best kill the rat fast. Note that a new rat spawns immediately upon the death of the first. The dead dire rat will not have disappeared before the new one spawns. Kill as many as you like, then close the game, and open the toolset and our demo module.

OK, dire rats are no problem. Let's make it more of a challenge. Place another waypoint next to the first waypoint and open its properties.



Figure 7 – Irene, the dire rat, and the fire beetle.

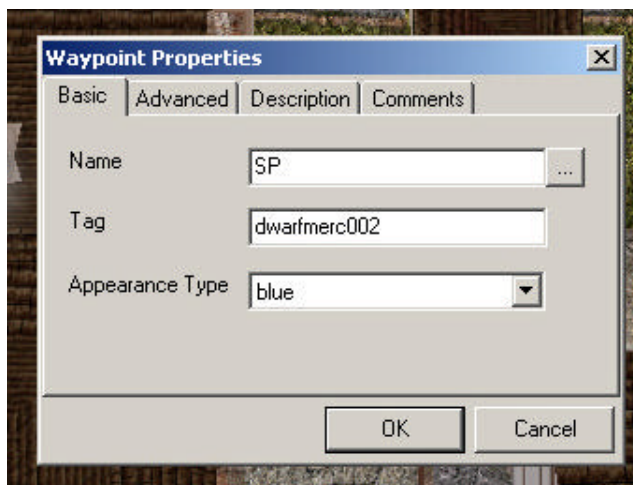


Figure 8 – Custom dwarf spawn waypoint.

Spawning Custom Creatures

Now you know how to create a simple spawn waypoint that will generate an unending supply of standard NWN dire rats and fire beetles. What if you want to spawn a custom creature?

Let's say you need a little help killing the never-ending supply of rats and beetles. Let's create a custom character to help. Go to the box on the right and select *Creatures > Standard > NPCs > Dwarves > Dwarf Mercenary*. Right

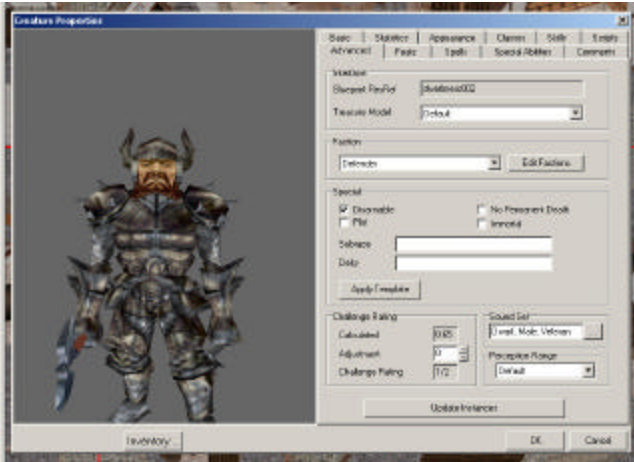


Figure 9 – Custom dwarf properties dialog box.

click, select *Edit Copy*. Change the dwarf's faction to *Defender*. Note the blueprint resref for the custom character and click *OK* to save it. My resref was *dwarfmerc002*.

Place a new waypoint half a tile in front of your PC starting point, between you and the rats. Put the *SP* flag in the name field of the waypoint and the resref of the custom creature in the tag field of the spawn waypoint as shown in Figure 8. Click *OK* and save the module. Start up the game.

Now your PC doesn't need to do much at all. Just stand and watch the dwarf kill the rats and beetles. Eventually the dwarf will succumb to his injuries, but no worry, another dwarf appears as soon as the first one dies.

The purpose of this chapter has been to show you how to select the type of creature you want to spawn at a waypoint. Later, I will show you how to spawn items, placeables, groups of creatures, and select the timing and number of creatures to be spawned.

To summarize, you create a spawn point by:

- 1) creating a waypoint,
- 2) changing the **name** of the waypoint to **SP**,
- 3) placing the **resref** of the standard or custom creature in the **tag** field of the waypoint.



Figure 10 – Dwarf at work.

Chapter 4 - Spawning Basics - SP, SN, and SA

In this chapter we will:

- learn the options associated with the SP option,
- learn how to change the number of creatures spawning with the SN option, and
- learn the options associated with the SA option.

SP[n]|Om – Spawn Point

The required SP flag has one optional parameter associated with it. An integer number greater than zero. SP1 is the default. Normally, with the flag set SP1, the area OnHeartBeat script poles every waypoint and evaluates if any actions need to be taken. This is done on every game heartbeat, which is about **every six seconds**. If the optional n parameter is set to something other than 1, the spawn processing will not be done on every game heartbeat for that spawn point, only every nth heartbeat. That means that if any spawning, despawning, random walk, or reassignment of activities needs to be done for that waypoint, it will not be done until the next nth heartbeat occurs.

Let's give this a try with our demo module.

- Open the *NESS Demo* module and the *City* area.
- Open the properties for the dire rat spawn waypoint.
- Change the name field from *SP* to *SP5*.
- Click OK to close.
- Open the fire beetle spawn waypoint and change the name to *SP10*.
- Click OK to save.
- Save the module.
- Open the demo for playing.

Note that when the dire rat and the fire beetle are killed, another creature is not immediately spawned. For the rat, the next opportunity for a spawn occurs at every fifth game heartbeat and for the fire beetle the next opportunity for a spawn occurs every tenth game heartbeat.

Now our rats will spawn every five game heartbeat and the fire beetles every tenth if we can kill them that fast. In effect, for the dire rat spawn waypoint, the heartbeat is every fifth game heartbeat and for the fire beetle spawn point it is every tenth game heartbeat. I will call this a **cycle** for the spawn point and it can be thought of as an **effective heartbeat** for the spawn point. In the Spawn Waypoint Generator (SWG) help file it is also called a cycle.

So, what good is this? You might say this is a great way to control the rate at which creatures are spawned. This is correct, it can be used for that, but NESS has a specific option for doing this, the SDn flag, that I will discuss later. The SP parameter is best used for game processing load control. If you have many spawn points in a game that rarely do anything after the initial spawn, then the SP parameter can be set to something like 10. This will reduce the processing of these spawn points to 10% of the default case of every game heartbeat.

The optional Om subflag is an offset to the first processing of the waypoint when the SP flag parameter is used. For example, if you have three waypoints set to SP3 you can use the Om subflag to stagger the processing of the waypoints. Set one waypoint to SP3 without the Om subflag, set one to SP3O1, and one to SP3O2. In this way the processing of the three flags will be done on different heartbeats.

All flags with parameters accept as many digits as you want on them, as long as it makes sense. The leading zeros shown in the *spawn__readme* file are not necessary either.

SNn|Mm – Spawn a Specific Number

The SNn flag determines the number, n, of creatures to maintain for that one spawn point. The default is SN1. The parameter n is required as the flag makes no sense without it. We have seen this in the examples above. Each time you killed a dire rat, another was spawned at the next cycle.

- Open the *NESS Demo* module and the *City* area.
- Open the properties for the dire rat spawn waypoint.
- Change the name field from *SP5* to *SP_SN3* as shown in Figure 11. **Note that the underscore character is used to separate primary flags in the name field.**
- Click *OK* to close.
- Leave the fire beetle spawn waypoint name at *SP10*.
- Save the module.
- Open the demo for playing.

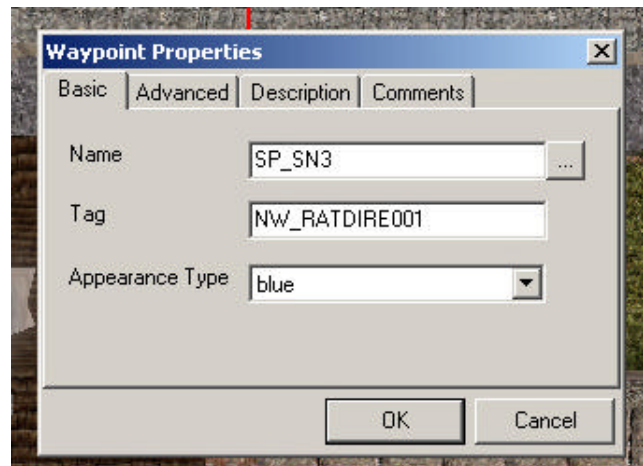


Figure 11 – Three dire rats.

What happened to the fire beetle? The spawning of the fire beetle did not change. Only one fire beetle per cycle was spawned. When it was killed it was not respawned until the next cycle, in this case every tenth game heartbeats. What happened to the rat? Now we have three rats. One rat was spawned every cycle, which was every game heartbeat, up to a maximum of three rats at any one time. If one was killed, then one was spawned at the next cycle, each game heartbeat. This may have been difficult to see if your dwarf was particularly good at killing rats. Let's try the demo with the dwarf and fire beetle turned off. Just change the name field in each waypoint by putting an *x* in front of the *SP*.

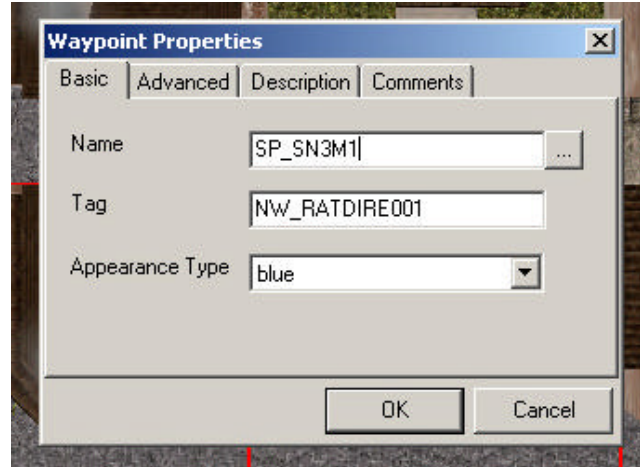


Figure 12 – Three dire rats, minimum one.

The optional flag *Mm*, where *m* is an integer less than the integer attached to the *SNn* flag, makes the waypoint spawn a random number of children between *m* and *n*. A setting of *SN3M1* will maintain between one and three creatures at the waypoint. Let's try it. Open the dire rat waypoint, change the *SN3* flag to *SN3M1* in the name field. Your name field will now be *SP_SN3M1* as shown in Figure 12. Save the module. Open the demo for playing.

In my demo, the PC is too far from the dire rats for them to notice her. They will only attack her if she attacks them. My PC is a ranger so she can attack from two tiles away. What happens? I almost always get three dire rats. I get two rats about 20% of the time. In addition, if you kill one dire rat, another one will almost always be spawned. The random number of creatures spawned is set at the initial spawning and only reset if all the creatures are killed during a cycle.

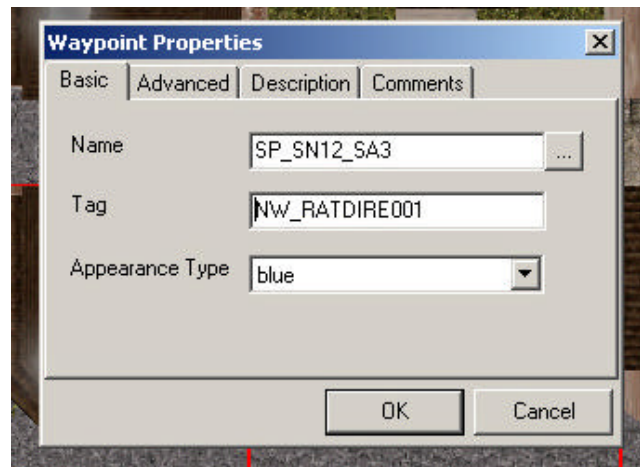


Figure 13 - Twelve dire rats, three at a time.

SA[n]Mm – Spawn at Once

The *SA[n]* flag allows you to specify the number of creatures to be spawned on each cycle. It is followed by an integer greater than one or no integer at all. Without the *SA[n]* flag, one creature will be spawned per cycle. Using the *SA* flag without any parameter, just *SA*, will spawn all the creatures at once. The default without the *SA* flag is equivalent to *SA1*, one creature per effective heartbeat. Since *SA1* is the default, there is

never a need to use the flag SA1. Use the SA[n] flag when you want more than one creature to be spawned at the waypoint per cycle.

Let's give it a test drive.

- Open the dire rat spawn waypoint.
- Change the name field to *SP_SN12_SA3*, as shown in Figure 13. This set of flags will give me a maximum of 12 rats, three being spawned at every cycle, which is every game heartbeat, or about six seconds.
- Save the module.
- Open the game for playing. Best be ready!

What happened? A bunch of rats, that's what. Each game heartbeat three more rats were spawned up to a maximum of twelve. If you killed a few, up to three more were spawned at each game heartbeat.

The optional subflag Mm will cause a random spawn of between n and m creatures at each cycle up to the maximum number of creatures. The flag SA4M2 will spawn randomly between two and four creatures on each cycle up to the maximum number of creatures.



Figure 14 – Irene and the twelve dire rats.

To summarize:

- The waypoint **name field** must start with the SP flag.
- An integer parameter can be attached to the SP flag to reduce the waypoint processing frequency.
- The flag SNn allows the waypoint to create any number of creatures. This is done by attaching an integer parameter to the SN flag greater than one, such as SN3.
- A random number of creatures can be spawned at a waypoint by using the Mn parameter attached to the SNn flag. For example, SN3M1 will produce from one to three creatures at the waypoint.
- The SAn flag allows the waypoint to spawn more than one creature per cycle, such as SA3 will spawn three creatures per cycle.
- Individual flags must be separated by the underscore character, i.e., SP_SN3.
- The spawn waypoint tag field must be the **resref** of the creature, NPC, placeable, or item to be spawned.

Chapter 5 - Spawning Day and Night – DY, DO, NO, and HR

In this chapter we will:

- learn how to spawn at specific times.

DYn|Tm – Spawn on a Specific Day of the Month

The DYn flag specifies the day of the Neverwinter month the creatures will spawn. The parameter n is an integer between 1 and 28. The creatures will spawn only on the one day specified. A flag DY3 will spawn creatures only on day 3 of the month. Creatures will spawn at hour 1 and despawn at hour 24. By using the optional Tm flag you can specify a period of days over which the creature will exist. A DY3T6 flag will spawn creatures on day 3 at hour 1 and remain until despawning on day 6 at hour 24.

HRn|Tm – Spawn at a Specific Hour

The HRn flag specifies the spawn will occur at hour n. HR12 will spawn the creature at game hour 12. The optional Tm flag indicates a despawn time. The flag HR12T20 will spawn the creatures at hour 12 and despawn them at hour 20 each day.

DO|D – Spawn Only During the Day

The DO flag specifies the spawn will occur only during the day, between the hours of 6 and 18. The optional flag D makes the spawned creature despawn at dusk, hour 18. Without the optional D flag the creatures will not despawn and remain until killed or despawned in another manner. The combined flag DOD will spawn the creatures at hour 6 and despawn them at hour 18. Assuming you are using the standard area settings of dawn at hour 6 and dusk at hour 18.

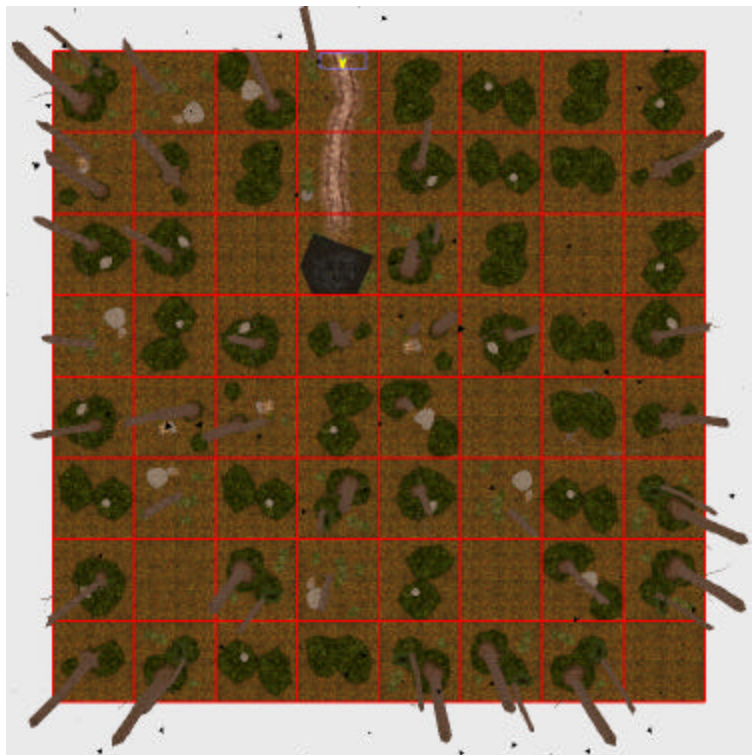


Figure 15 – Forest area.

NO|D – Spawn Only During the Night

The NO flag is the compliment of the DO flag. With the NO flag the creatures will spawn only between the hours 18 and 6. The optional flag D makes the spawned creature despawn at dawn, hour 6. The combined flag NOD will spawn the creatures at hour 18 and despawn them at hour 6. Again, assuming you are using the standard area settings of dawn at hour 6 and dusk at hour 18.

Let's add to our module and create an example to illustrate the use of these flags.

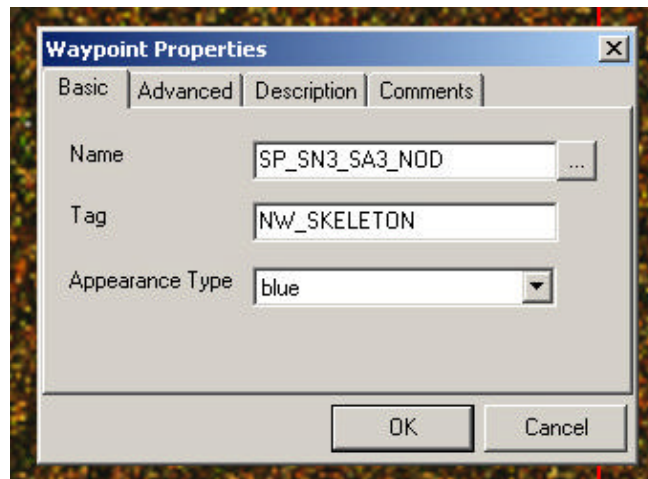


Figure 16 – Three skeletons night only.

- In the *City* area add a transition trigger at the end of the street where it exits on the south edge of the.
- Paint the transition trigger completely across the street. Open the trigger properties dialog box and give it the tag *city_into_forest*, click OK.
- Add a new area using the Area Wizard.
- Name the new area *Forest*, select the Forest tileset, and make it 8 x 8 tiles.
- Open the area properties and add the *spawn_sample_hb* script to the *OnHeartBeat* event.
- Add a two tile length of road coming down from the center north edge.
- Where the road meets the edge paint another trigger; give it the tag *forest_into_city*.
- Right click on the trigger and open the trigger properties. In the transitions tab, attach it to the *city_into_forest* trigger of the area *Town*. Your finished area should look like Figure 15.

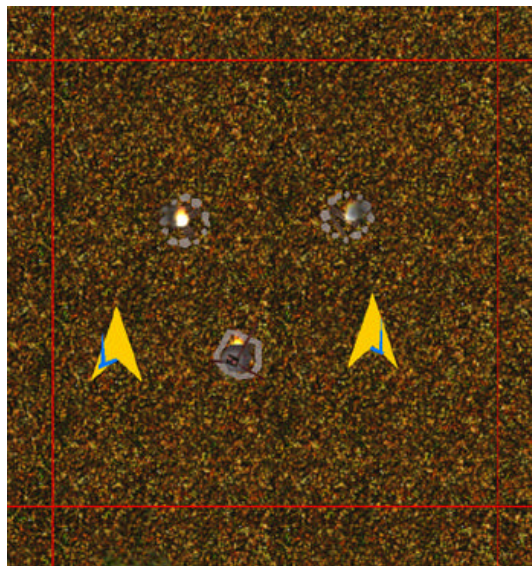


Figure 17 – Three skeletons.

OK, now we have our new Forest area. Place three campfires close to each other in the southwest corner of the area in the middle of the second tile up and second tile from the left. Add a waypoint next to the fires. In the name field of the waypoint put *SP_SN3_SA3_NOD* and in the tag field *nw_skeleton*, as shown

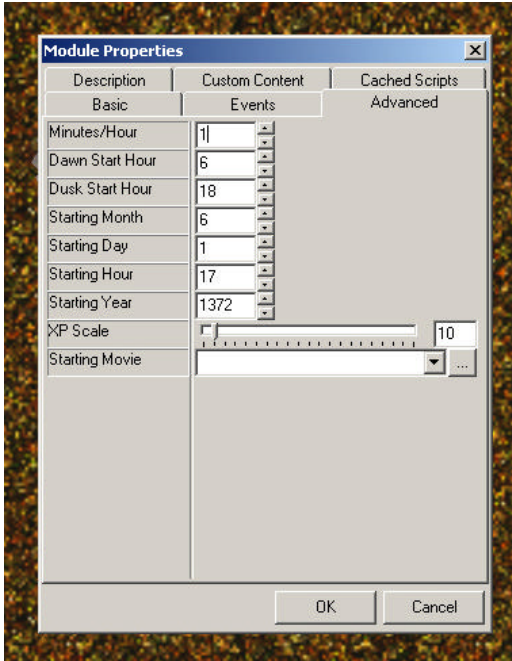


Figure 18 – Module properties.

another waypoint and have a skeleton chief show up at hour 21 to check up on the skeleton warriors and then disappear at hour 24. The waypoint name for this would be *SP_HR21D24* and the tag *nw_skelchief*.

To summarize:

- spawn a creature on a specific day of the Neverwinter calendar with the DY flag,
- spawn a creature on specific days of the calendar with DY flag,
- spawn a creature at a specific hour of the day with the HR flag,
- spawn during a period of the day with the flag HR,
- spawn only during the day with the DO flag,
- spawn only during the day and despawn at dusk with the DOD flag,
- spawn only during the night with the NO flag,
- spawn only during the night and despawn at dawn with the NOD flag.

in Figure 16. Place another waypoint on the other side of the campfire. In the name field of the waypoint put *SP_SN3_SA3_DOD* and in the tag field *nw_orca*. Your finished camp should look like Figure 17. Place the PC start point in the upper left facing the bottom. Open the module properties by selecting *Edit > Module Properties*. Select the *Advanced* tab and change the start time of the module to hour 17 and the rate to 1 minute per hour, as shown in Figure 18. Close the *Area Properties* dialog box, save the module, and open it for play.

Move south until you can just see the campfires and orcs. You should be out of the orcs perception range. They're standing around doing nothing. Wait for a minute and they will despawn at hour 18. At the same time, the skeletons spawn. We could add

Chapter 6 - Patrolling – PR

In this chapter we will:

- learn how to get creatures to go on patrol

PRn|Tn|C – Spawn a Creature to Patrol a Route

Standing around doing nothing is not what you want! OK, let's put those creatures on patrol. Use the PRn|Tn flag to set up a patrol route. The PR flag must be followed by an integer parameter between 1 and 99. This number indicates the **patrol route number** and is used in the waypoint tags that define the patrol route. The optional T flag determines the way the route will be patrolled as follows:

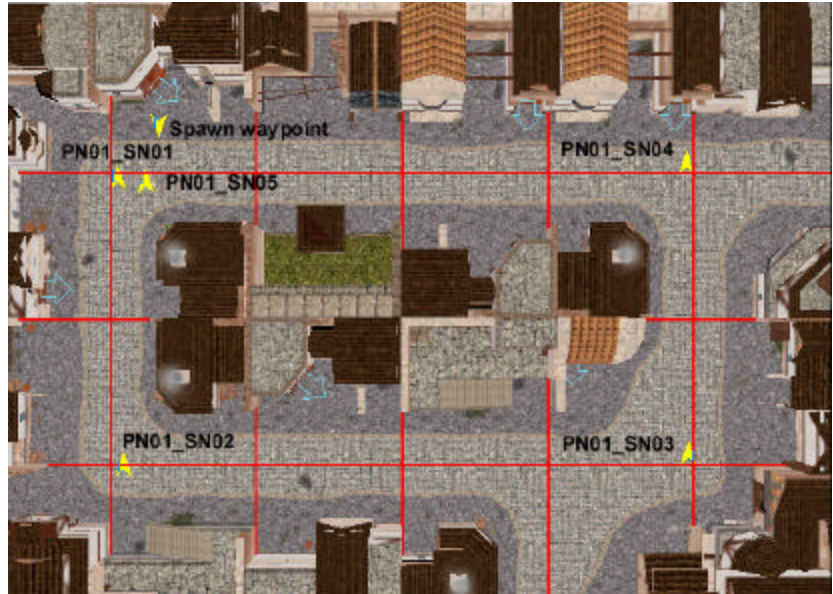


Figure 19 – Make some patrol waypoints.

- T0 = Ascending, descending order, repeat
- T1 = Circular, repeat
- T2 = Random
- T3 = Sequentially, despawn

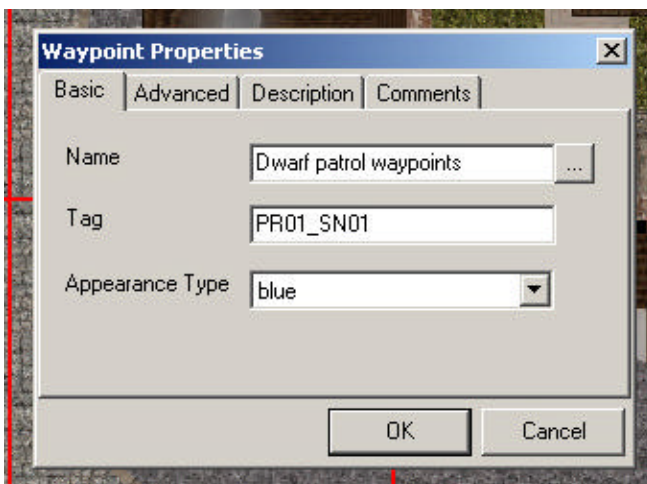


Figure 20 – Dwarf patrol waypoint one.

The route of the patrol is defined by waypoints. The name field of the waypoint is not used and may be anything. The tag field of the waypoint must start with PRn_SNm where n is the **patrol route number** and m is **stop number**. M must be sequential and continuous beginning with any integer number between 0 and 99. The C subflag tells the creature to start the patrol at the nearest patrol route waypoint rather than the default, which is to start with the first. The C subflag

has no effect when used with subflag T2.

Let's do an example. The first patrol route waypoint is shown in Figure 20. Lay out five waypoints, PR01_SN1 through PR01_SN5, around the block in the northwest corner of the Town area as shown in Figure 19. Place the spawn waypoint on the sidewalk north of waypoint PR01_SN05 as shown in Figure 19. Put *SP_PRO1T0* in the name field and the **resref** of our dwarf mercenary in the tag field, *dwarfmerc001*, in my case. Your spawn waypoint properties should look like Figure 21. The flags we have set should spawn one dwarf. The dwarf will follow patrol route 01 in ascending order until reaching the last patrol waypoint, number five, then reverse and follow the patrol route in descending order, then repeat. Put the PC start point in the middle of the street on the west. Save the module and open it for play.

The one dwarf spawned and set off on patrol. The dwarf followed the patrol waypoints in ascending sequential order until the last one, then returned in descending sequential order, repeat. Just what we predicted. Exit the game, open the toolkit.

Now change the optional flag parameter, T0, on the patrol flag to T1. Your spawn waypoint flags should now be *SP_PRO1T1*. The flags we have set should spawn one dwarf. The dwarf will follow patrol route 01 sequentially until reaching the last patrol waypoint, number five, and then begin again. Save the module and open it for play.

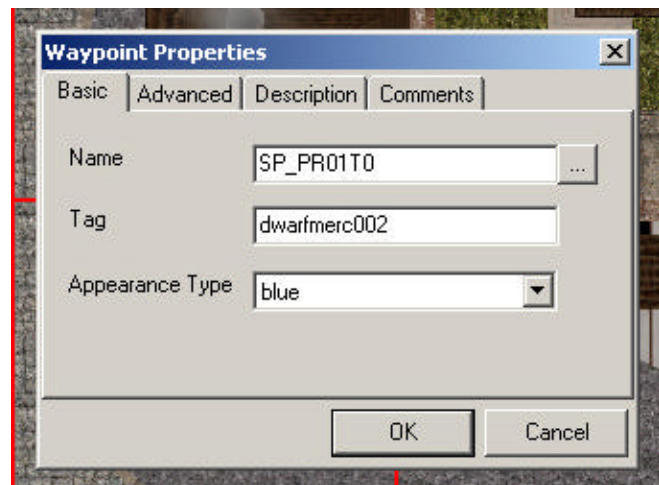


Figure 21 – Dwarf on patrol.

The dwarf spawned and set off on patrol. The dwarf followed the patrol waypoints in ascending sequential order. When he reached the last patrol waypoint, he began the patrol again from waypoint one in ascending order, i.e., a circular route. Just what we predicted. Exit the game, open the toolkit.

Now change the optional flag parameter, T1, on the patrol flag to T2, and add the SN3 flag to get three dwarfs spawned. Your spawn waypoint flags should now be *SP_SN3_PRO1T2*. Now, after the dwarves spawn, they should randomly select the next patrol waypoint to which they will proceed. Each will follow his own route from one waypoint to the next. Upon reaching a patrol waypoint, each will select the next patrol waypoint randomly. Save the module and open it for play.

The three dwarfs spawn one after the other on each game heartbeat and set off on patrol. They randomly select the next patrol waypoint to which to proceed next. Just what we predicted, again. Exit the game, open the toolkit.

Now change the optional flag parameter, T2, on the patrol flag to T3. Your waypoint flags should be *SP_SN3_PR0IT3*. This time the dwarves should follow the patrol waypoints in ascending sequential order. When they reach the last patrol waypoint, they will despawn. Save the module and open it for play.

One dwarf spawns immediately and he starts around the waypoint patrol route. Next game heartbeat another spawns and then another. When dwarf one reaches waypoint five he stops, and a moment later despawns. At the next cycle another dwarf spawns and begins his rounds. Then dwarf two arrives at waypoint five, stops and despawns. At the next cycle a new dwarf spawns and starts on patrol. When you've seen enough, exit the game. In this example the three dwarfs spawn one after the other without delay.

The format of the patrol route waypoint tag is:

Route Tag: PR00_SN00_PS000_RN_FC_DO_NO_SC000

Where:

PR = route number, 00 to 99.
SN = stop number, 00 to 99.
PS = pause 000 to 999 seconds at stop.
RN = run to stop.
SF = face the waypoint direction.
DO = only stop here during the day.
NO = only stop here during the night.
SC = run script 000 to 999, scripts are defined in file *spawn_sc_patrol*.

The basic patrol route waypoint is shown in Figure 20. The six additional flags may be used to control the actions of the patroller. Let's try a few of them. Change the optional flag parameter, T3, on the spawn waypoint patrol flag to T1 so that our patroller takes a circular route and remove the SN3 flag so we only get one patroller. Our waypoint tag should now be *SP_PR0IT1*. Open the properties of the first patrol waypoint and add the PS flag so our patroller pauses for 15 seconds. Your waypoint flag should be *PR01_SN01_PS15*. Open the properties of the second patrol waypoint and add the SF flag so our patroller will face the direction of the waypoint. Your waypoint flag should be *PR01_SN02_SF*. Open the properties of the third patrol waypoint and add the DO flag so our patroller will stop here only during the day. Your waypoint flag should be *PR01_SN03_DO*. Open the properties of the fourth patrol waypoint and add the NO flag so our patroller will stop here only during the night. Your waypoint flag should be *PR01_SN04_NO*. The use of the SC flag will be discussed in the next section. Save the module and open it for play.

See the patroller delay, face, run, stop only during the day, stop only during the night. Exit the game.

Turning Off PR

Add an ID to the spawn point. From within another script, search for the ID number and use the following command to deactivate the PR:

```
SetLocalInt(oSpawn, "f_PatrolRoute", -1);
```

Extra Credit – Making the Patroller do Something

OK, let's make the patroller do something a little more interesting. There are two simple methods to do this that I will demonstrate. There are others, but I will not go into them here. One method is to use the optional SC flag on the patrol route waypoint to have the patroller run a script and the other is to use the OnPerceive event.

The optional SCn flag is placed in the tag field of the patrol waypoint where the scripted actions are to occur. The parameter n is an index used in the *spawn_cc_patrol* script to

```
//
// Only Make Modifications Between These Lines
// -----

// Script 00
if (nPatrolScript == 0)
{
    ActionDoCommand(SpeakString("Example!"));
}
//

// Turn Off Lights
if (nPatrolScript == 7)
{
    object oLight = GetNearestObjectByTag("Light", oStop);
    if ((GetIsDay() == TRUE && GetPlaceableIllumination(oLight) == TRUE)
        || (GetIsNight() == TRUE && GetPlaceableIllumination(oLight) == FALSE))
    {
        ActionDoCommand(DoPlaceableObjectAction(oLight, PLACEABLE_ACTION_USE));
    }
}
//

// Trap-Setting Rogue
if (nPatrolScript == 1)
{
    object oTrap = GetNearestTrapToObject();
    if (oTrap == OBJECT_INVALID || GetDistanceToObject(oTrap) > 5.0)
    {
        // Create a Trap Kit
        object oTrapKit CreateItemOnObject("NW_IT_TRAP001", OBJECT_SELF, 1);
        // Set Trap
        SignalEvent(GetModule(), EventActivateItem(oTrapKit,
        GetLocation(OBJECT_SELF)));
    }
}
//

// -----
// Only Make Modifications Between These Lines
//
```

Figure 22 – script_cc_patrol.

```

//
// Only Make Modifications Between These Lines
// -----
.
.
.
// dwarf guard checks door
if (nPatrolScript == 99)
{
    int nRoll = d3(1);
    if (nRoll == 1)
    {
        object oDoor = GetNearestObject(OBJECT_TYPE_DOOR, oStop, d3(1));
        float fDist = GetDistanceBetween(oStop, oDoor);
        if (oDoor != OBJECT_INVALID && GetDistanceBetween(oStop, oDoor) < 16.0)
        {
            // goto door and examine it
            ActionSpeakString("Think I'll check that door over there.");
            ActionMoveToObject(oDoor, FALSE, 0.5);
            ActionPlayAnimation(ANIMATION_LOOPING_GET_MID, 1.0, 3.0);
            switch (d3(1))
            {
                case 1:
                    ActionSpeakString("Dee dum, dee dum, looks OK");
                    break;
                case 2:
                    ActionDoCommand(SpeakString("Yup, seems to be locked real good."));
                    break;
                case 3:
                    ActionSpeakString("Hay, you in the house, lock you doors!");
                    break;
            }
        }
    }
}
// -----
// Only Make Modifications Between These Lines
//

```

Figure 23 – Check the door script.

locate the custom scripted actions to perform at the stop. It is an integer greater than 0. Remove the PS, SF, DO, and NO flags we just placed on the patrol waypoints. They should all now be simply PR01_SN0x, where x is the number of the patrol stop. Open patrol stop number 3 and add the SC99 flag; the waypoint tag should now be *PR01_SN03_SC99*. We have just set the patrol stop to run a script when the patroller gets to this stop.

Open the file *spawn_sc_patrol*. This is the script that will be executed when the SCn flag is added to the patrol stop. In it you will find the code section listed in Figure 22. This is where we will put the script we create that will be run when the patroller reaches the patrol stop. Three scripted actions have already been included. The first does nothing except makes the patroller speak “Example.” The second has the patroller turn a light off or on, and the third has the patroller set a trap. In our case the parameter 99, following the optional SC flag, is the value of *nPatrolScript* when the *spawn_sc_patrol* file is executed for our patrol stop. We will add a section to the *spawn_sc_patrol* file in the format:

```
// dwarf stop
```

```

if (nPatrolScript == 99)
{
    .
    .
}

```

What do we want our patroller to do? Our dwarf is a guard, so let's have him do something a patrolling guard would do - check the doors nearby to make sure they are locked. We will be using patrol stop three for this. Make sure you have placed at least three doors in all the doorways around this stop. We will script the action so the patroller selects from one of three nearby doors to check. The script will perform the following functions:

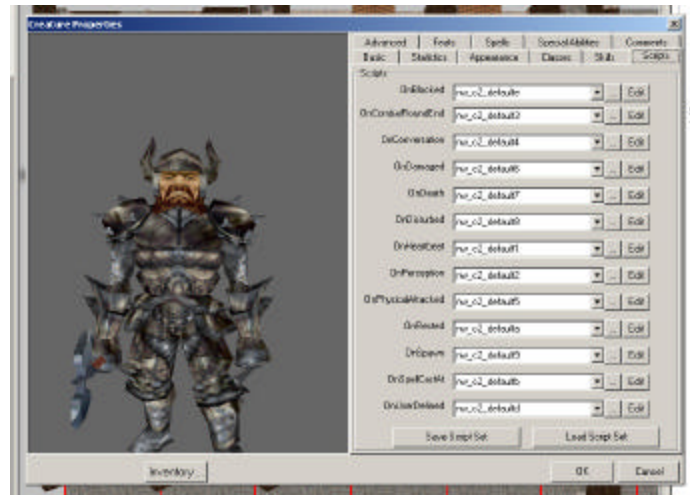


Figure 24 – Custom dwarf scripts dialog box.

- randomly determine if the actions will be executed at all, a chance will do,
- randomly select from one of the three nearest doors to check,
- if the door is within a specified distance then go to the door and,
- randomly select from three expressions to say while checking the door.

The logic executed will be:

```

if ( chance = true) then {
    select one of three nearest doors
    if (the selected door is within 15 meters) then {
        move to selected door
        perform animation on door
        select one of three expressions to utter
        utter selected expression }}

```

The script in Figure 23 will meet our needs. Several debugging lines have been included and commented out. You can ignore them. Place this script in the `spawn_sc_patrol` file after the trap setting script and before the line at the bottom that says *Only Make Modifications Between These Lines* as shown in Figure 23.

When you make modifications to any of the spawn files, you must recompile the `Spawn()` function. This function is called from the `spawn_sample_hb` script. Open the `spawn_sample_hb` script and compile the script. Save the module, open it for play.

When the game starts, one dwarf is spawned and starts on patrol. Follow him to patrol waypoint three. There is a one third chance he will perform the action we scripted. Wait until the dwarf on patrol performs the action. Note that after finishing the scripted

actions, the dwarf returns to his patrol route at the next cycle. If the cycle has been set for several game heartbeats using the SPn parameter, then the patroller may stand around for several seconds before resuming his patrol. If the dwarf does not seem to be doing the scripted actions, it may be that none of the doors are within the specified range.

Our patroller is now doing something useful. We can also set this flag, SC99, on any other patrol waypoint. Or we can use the existing script to light street lamps or equip a torch.

```

//:////////////////////
//: Default: On User Defined
//: NW_C2_DEFAULTD
//: Copyright (c) 2002 Bioware Corp.
//:////////////////////
/*
    Determines the course of action to be taken
    on a user defined event.
*/
//:////////////////////
//: Created By: Don Moar
//: Created On: April 28, 2002
//:////////////////////
void main()
{
    // enter desired behavior here
    return;
}

```

Figure 25 – OnUserDefine default NWN script.

Now let's try something using the *OnPerceive* event for our patroller. Open the custom dwarf properties dialog box and select the *Script* tab, as shown in Figure 24. Select the *Edit* button on the *OnSpawn* event to open the script editor. On line 204, or there about, find the function:

```
// SetSpawnInCondition(NW_FLAG_PERCIEVE_EVENT);
```

Remove the two slashes, //, from in front of the function to enable it. Select the *Save as* button and give the new script the name *dwarf_onspawn*. What we have done here is to enable the creature to fire an event response. When the creature perceives another object it sets a variable that can be obtained by the *GetUserDefinedEventNumber()* function. For perceived events this variable value will be 1002. To use the perceived event we test for the value in the *OnUserDefined* event handler for the creature and script a response. So let's do that. Click on the *Edit* button for the *OnUserDefined* event and you will see the script shown in Figure 25. This is the default script for this event; it does nothing. Replace it with the script shown in Figure 27. Select the *Save as* button and give it the name *dwarf_userdefine*. This script is a simplification of Spikes Intelligent Militia script. The replaced script obtains the value of the *GetUserDefinedEventNumber()* function and then acts according to the value. For all values except value 1002 it does nothing. For value 1002 it determines what was perceived, if it was a PC it determines if the PC was perceived by being seen. If so it determines if the PC is carrying a weapon. If all the above is true, then the script makes the NPC approach the PC and engage in a

conversation defined in the conversation file of the creature NPC. Select the *Basic* tab of the custom dwarf properties and create the conversation file shown in Figure 26.

We've made some changes to the spawn configuration files so you need to recompile. Save the module, open the game for play. Equip a weapon and wait for the dwarf to come around the corner. As soon as he rounds the corner he perceives you and your weapon. You're in trouble now! You could modify this to take into account how you respond to the guard and how many times he has already warned you. But, that's a subject for another tutorial.

To summarize:

- to get your spawned creature to patrol a route, use the PRn flag,
- there must be a set of waypoints that define the route,
- the route waypoints must have the tag PRn_SNm in each one, and
- you can use the optional flags to control the actions of the patrolling creature.

```
[Owner] - You are not permitted to carry weapons in the city. You must stow all weapons while
within the city or you will be fined or placed in jail. Do you understand?
    Yes, Yes, I'll put it away.
    [Owner] - Good! Don't let me see that weapon out again! Be warned! The city guards are
on the lookout for brigands, criminals, and any other rascals out there who think they can disrupt
the peace of our fine city. Now get!
    OK, don't bust a gut over such a small thing.
    [Owner] - Good! Don't let me see that weapon out again! Be warned! The city guards are
on the lookout for brigands, criminals, and any other rascals out there who think they can disrupt
the peace of our fine city. Now get!
    Really! Do I have to? I won't hurt anyone.
    [Owner] - <StartHighlight>[Yes! DISARM NOW!]</Start>
    OK, don't bust a gut over such a small thing.
    Bugger off you wimp!
    [Owner] - OK!, I warned you! It's off to the dungeons for you.
```

Figure 26 – Dwarf patroller conversation.

```

//:////////////////////
//: Default: On User Defined for dwarf guard
//: dwarf_onuserdefine
//:////////////////////
/* Determines the course of action to be taken
   on ser defined event on perceive. */
//:////////////////////
void main()
{
int nUser = GetUserDefinedEventNumber();
  switch (nUser)
  {
    //case 1001: // on Heartbeat Event
    //break;

    case 1002: // on Perception Event
      object oPC = GetLastPerceived();
      //check to see if I actually saw a PC
      if (GetLastPerceptionSeen() && GetIsPC(oPC))
      {
        object oItem1 = GetItemInSlot(INVENTORY_SLOT_LEFTHAND,oPC);
        object oItem2 = GetItemInSlot(INVENTORY_SLOT_RIFTHAND,oPC);
        int nItem1 = GetBaseItemType(oItem1);
        int nItem2 = GetBaseItemType(oItem2);
        //SpeakString("nTreat= "+IntToString(nTreat), TALKVOLUME_SHOUT);

        // check for weapons drawn
        if ((nItem2 != 256) && (nItem1 != 256 || nItem1 != 14 || nItem1 != 15 || nItem1 !=
56 || nItem1 != 57))
        {
          int nWarn = GetLocalInt(oPC, "lower_weapon_warning");
          nWarn++;
          SetLocalInt(oPC, "lower_weapon_warning", nWarn);
          // clear NPC and PC actions
          ClearAllActions();
          AssignCommand(oPC, ClearAllActions());
          // speak as guard approaches
          ActionSpeakString("You there! Halt!");
          // move to PC and face the PC
          ActionMoveToObject(oPC);
          ActionDoCommand(SetFacingPoint(GetPosition(oPC)));
          // animate taunt at PC
          ActionPlayAnimation(ANIMATION_FIREFORGET_TAUNT);
          ActionStartConversation(oPC);
        }
      }
      break; // end Perception Event
    //case 1003:// on End Combat Round Event
    //break;
    //case 1004:// on Dialog Event
    //break;
    //case 1005:// on Attacked Event
    //break;
    //case 1006:// on Damaged Event
    //break;
    //case 1007:// on Death Event
    //break;
    //case 1008:// on Disturbed Event
    //break;
  }
}

```

Figure 27 – OnUserDefine script for dwarf.

Chapter 7 - Walking and Delaying – RW, SD, and IS

In this chapter we will:

- learn how to get those creatures to do more than stand around, and
- learn how to delay spawns.

RW|Rn – Random Walk

So, got those creatures spawning, but they just stand around doing nothing. One of the easiest ways to liven up the place is to add the random walk flag RW. Go back to our camp in the southwest corner of the Forest area. Open the properties for the skeleton and orc waypoints and add *RW* to the flags. The skeleton waypoint properties will look like those shown in Figure 28. Try the game. See the creatures wander around. That's all there is to the RW flag.

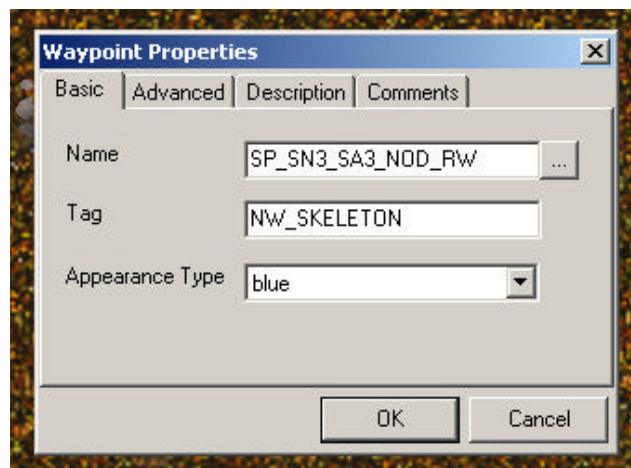


Figure 28 – Three skeletons walking random.

Turning Off RW

Add an ID to the spawn point. From within another script, search for the ID number and use the following command to deactivate the RW:

```
SetLocalInt(oSpawn, "f_RandomWalk", FALSE);
```

RH|Dn – Return Home After Action

So your creature is randomly walking around now. And walking, and walking, and walking. Pretty soon it's all the way on the other side of the area, not where you want it to be! Or, it attacks you (heaven forbid) and you run, and run, and run. Finally it loses your trail and stops. Again, it is on the other side of the area, not where you want it. Add the RHn flag and get it to return to the spawn location if it gets too far away. The parameter *n* is the number of meters the creature can wander before being forced to return to the spawn location. Notice I say **spawn location**, not spawn point. If you have used the SL flag to have the creature spawn at another location (discussed in the next chapter) then it will return to the location at which it was spawned, not the spawn waypoint. Also, it will return only after it has completed any actions in which it is engaged. It will chase you and engage in combat without returning. Only after completing any actions will it return home.

SDn|Mm|P – Delay Spawning

What if you wanted three dwarfs on patrol, but more spaced out? Change the flags in the dwarf spawn point to *SP3_SN3_PR0IT3*. Now there will be a pause of three game heartbeats between spawns. Save the module and start the game. The first dwarf spawns and then three game heartbeats later the second and then the third. When the first dwarf reaches patrol point six he must wait until the beginning of the cycle to despawn. That's because you have set the spawn point to have an effective heartbeat of 18 seconds. Nothing happens until the next processing cycle of the waypoint, every 18 seconds.

The SD flag will help you get what you want here. The parameter on the SD flag is a delay in *minutes* from the time the creature would normally have been spawned. If the creature would have been immediately spawned, that spawning will be delayed 1 minute with an SD1 flag added. Set the spawnpoint flags to *SP_SN3_SD1_PR0IT3*. These flags will spawn a maximum of three dwarves. The time between spawns will be one minute, but the processing cycle will be every game heartbeat. Consequently, no hanging around waiting to despawn. The one drawback to this is that the first spawning will be one minute after the area opens. This can be solved with the use of the IS flag discussed below.

The optional Mm subflag will set a minimum delay in minutes, in which case the n parameter sets a maximum delay before respawning and the respawn will occur randomly between minimum m minutes and maximum n minutes.

The optional P subflag sets up a regular schedule for respawning. The flag SD10P will schedule the spawn point to respawn if necessary every ten minutes. As an example, you want four skeletons wandering around a graveyard only at night. You want the skeletons to spawn all at once and then respawn as necessary every four game hours regardless of when they are killed. Your module clock rate is set at the default of 2 minutes per hour. You use the flags *SP_SN4_SA_RW_RH15_NOD_SD8_IS1D0* (the IS1D0 flag causes an initial immediate spawn and will be discussed below). You think this will spawn 4 skeletons immediately at hour 18. If any are killed during the four game hours before hour 22, then they will be spawned at hour 22. That is not what happens! The four skeletons are spawned at hour 18 as you expected. They wander around until hour 20 when your PC arrives, kills two and runs off. At hour 22 you expect the two killed skeletons to be replaced, but nothing happens. At hour 24 two skeletons are spawned. This is not what you expected or wanted. The SD flag without the optional P subflag causes the respawn of the two skeletons to be delayed 8 minutes from when the respawn would have occurred without the SD flag. The respawn would have occurred immediately after they were killed at hour 20, therefore the delayed respawn will now occur at hour 24. All you need to do to fix this is change your SD8 flag to SD8P, *SP_SN4_SA_RW_RH15_NOD_SD8P_IS1D0*. This will create a regular periodic replacement spawn. Now, regardless of when or how many of the skeletons are killed, they will all be replaced, if necessary, every eight minutes or four game hours.

ISn|Dm – Initial Spawning State

The ISn flag sets the initial state of the spawn point. By default the initial state of the spawn point is active. IS1 sets the initial state of the spawn point to be active. So, what good is this, you ask since this is the default? The IS1 flag alone is not much use, but used in conjunction with the optional subflag Dm it does have a useful purpose. The optional subflag sets the delay of the initial spawn in minutes. IS1D1 sets the spawn to active and delays the first spawn for one minute. If you have used the SD flag to delay the respawning of creatures after you have killed them, you will have also delayed the initial spawn of the creature.

Let's go back to the example we used in Chapter 5 - Spawning Day and Night – DY, DO, NO, and HR, where we created a spawn of three skeletons at night only using the flags SP_SN3_SA3_NOD. In that case, if one of the skeletons is killed another will be spawned at the next cycle. Let's say we don't want that, we want only three skeletons spawned at dusk and that's all there will be for the rest of the night, no respawns if some get killed. We could use the deactivate flag DS, but we have not gotten to that chapter yet so let's use the SD flag to those above to get SP_SN3_SA3_NOD_SD12. This will delay the respawning of any skeletons killed until day hours, when no spawning will occur because of the NOD flag (assuming you still have the module set to 1 real minute per game hour). Do the same for the orc during the day and you will get a set number of creatures around the camp fires without replacements. They will be replaced at the beginning of the next night or day. "Great," you say, "but the first time I go into the area there are no creatures." That's because the SD12 flag postponed the initial spawn for 12 minutes. Solve this by adding the flag IS1D0 also. This will set the spawn point to be initially active and delay the first spawn for zero minutes, i.e., immediate spawning. Now your flags look like SP_SN3_SA3_NOD_SD12_IS1D0.

IS0 sets the initial state of the spawn point to inactive. Nothing will happen until the spawn point is activated in some manner via a script. In the script, use the function NESS_ActivateSpawn(object oSpwn) or NESS_ActivateSpawnByID(int nSpawnID, object oArea). You can also activate the spawn from the custom check script spawn_chk_custom.

To summarize:

- to get your spawned creature to randomly walk around, add the RW flag,
- to get your spawned creature to return to the spawn location if they get too far away, add the RHn flag,
- to get the spawn point to delay spawns from the normal time of spawning, add the SDn flag,
- to get the spawn point to periodically replace killed creatures, use the P subflag on the SDn flag, i.e., SDnP flag, and
- to control the initial state of the spawn and the initial delay of the spawn, add the ISnDn flag.

Chapter 8 - Spawn Placement – SR, SL, SF, ST, and RS

In this chapter we will:

- learn how to spawn within a radius,
- learn how to spawn at another location,
- learn how to spawn a creature facing a specific direction, and
- learn how to spawn a sitting creature.

SRn|Mm|P – Spawn Within a Radius

The SRn flag sets a radius of n meters within which to spawn the creature or placeable. The spawn will occur anywhere within n meters of the spawn location. The parameter n is an integer greater than zero. The optional subflag Mm sets a minimum radius, so that the spawn will be between m and n meters. The m parameter is an integer greater than 1 up to the value of n. So SR10 will spawn a creature somewhere within a radius of 10 meters of the spawn location. The flag SR10M8 will spawn a creature somewhere in an annulus 8 to 10 meters from the spawn location. The flag SR10M10 will spawn the creature somewhere on the circumference of a circle 10 meters in diameter centered on the spawn location. The optional P subflag specifies the spawn will occur centered on a random PC in the area. The flags SR1M1P will cause a creature to spawn somewhere around, and one meter from, an unsuspecting PC in the area.

SLn|Rm – Spawn at Another Location

The SLn flag will cause the spawn to occur at another location. For clarity sake I will call this the spawn location, as opposed to the spawn point, which is the waypoint that defines the spawn. The parameter n is used in the tag field of the spawn location waypoint to identify it as the target of the spawn. The spawn location is a waypoint with any name and a tag field of SLn. So, the flag SL5 in the flag set will cause the spawn to occur at a waypoint with the tag SL5, and a name field of anything. The optional Rm subflag causes the spawn to occur at a random waypoint with a tag field of SLn through SLm. The flag SL5R10 will cause a random spawn to occur at one of the waypoints with tags between SL5 and SL10. Six waypoints must exist with tags of SL5 through SL10 for this to happen.

Simple example: 1) Open the *Town* area. 2) Place two waypoints in the street near the exit to the east. 3) Give them the tags SL01 and SL02, the name fields are not used. 4) Place a waypoint two tiles south and put in the name field SP_SN2_SL01R02 and in the tag field *nw_rat*. 5) Place the PC starting point just south of the spawn point. 6) Save the module. 7) Open it for play. Rats will start spawning from the two waypoints, SL01 and SL02, randomly. Kill them as they spawn to get more. See them spawn randomly at two spawn locations. When you've had enough, close the game, open the Toolset.

SF – Spawn Facing

OK, what does it mean to spawn facing you ask? Usually the spawn system randomly faces the spawned child. But sometimes we really want the spawned child to be facing a particular direction. Want to spawn a treasure chest, sure, but you don't want it to be facing the wall of the cave! So there is the SF flag. How does it work? Not logically, but consistently. Add the SF flag to a spawn point for a creature and the creature will face the direction of the waypoint flag pointer. Add the SF flag to a container and it will open in the direction of the back of the waypoint flag pointer!

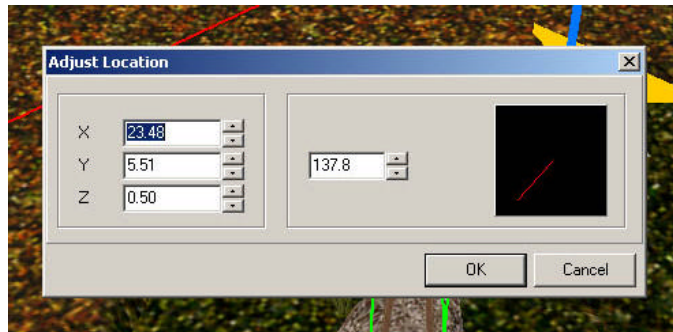


Figure 29 – Adjust Location dialog box.

ST – Spawn Sitting

Add the ST flag to a creature's flag set and it will look for and sit in the nearest sitable object with the tag name *Seat*. That's simple. Put a chair somewhere near the spawned creature, change its tag to *Seat*, and the creature will sit on it.



Figure 30 – Not so invisible chair.

What about all those useless chairs, stools, and benches that are built in features of some tile sets? No problem. Select a stool or chair, set its appearance as an invisible object, give it the tag *Seat*, and save it as a custom placeable. Now place the custom, invisible stool in the tileset chair or bench where you want the spawned creature to sit. Voilà! Previously useless bench, stool, or chair is now sitable. Having trouble getting the invisible chair to stay where you want it. Right click on the object in the tile set, select *Adjust Location* and set the location you want.

Deactivate all the existing spawn points in the Forest area by putting an x before the



Figure 31 – Invisible chair.

SP. Make a waypoint in the Forest area in the southwest corner, near one of those rocks that are about chair height, with the flag set SP_ST and the tag field *nw_comfemale*. Take a chair object and place it in the rock. Change the tag of the chair to *Seat*. Use the *Adjust Location* dialog box to set the chair at $z = 0.0$ if necessary as shown in Figure 29. Do not click on the chair again or it will pop out of the rock and you will have to reset it. Change the appearance of the chair to Invisible Object. Save the module, open it for play. Go to the location of your commoner female and she will be sitting on the rock. You can use this technique with any of the tileset benches and chairs.

RSn – Random Spawn

The RS flag creates a random spawn n% chance of occurring. The flag RS50 will give the spawn point a 50% chance of spawning a creature when a spawn is suppose to occur. That's all.

To summarize:

- to get the creatures to spawn within a circle, annulus, or on a circumference, use the SR flag,
- to get the creature to spawn at another location, use the SL flag,
- to get the creature or placeable to face a specific direction, use the SF flag,
- to get the creature to sit, use the ST flag, and
- to set a probability that a spawn will occur at all, use the RS flag.

Chapter 9 - Entrance and Exits – EE and EX

In this chapter we will:

- learn how to spawn and despawn at another location.

EE_n|R_m – Enter and Exit at Waypoint EE_n

The EE_n flag is similar to the SL_n flag. Using the EE_n flag in the flag set will cause the creature to spawn and/or despawn at the location of a waypoint with the tag EE_n. The flags SP_EE01 will cause the creature to spawn at a waypoint with the tag EE01. Using the optional R_m subflag will cause the creature to select a random waypoint with tags EE_n through EE_m for spawning and despawning. The creature will not necessarily use the same waypoint for both. The flags SP_EE01R03 will have the creature spawn and/or despawn at one of the randomly selected waypoints with tags EE01, EE02, or EE03. Of course, the creature must have a reason to despawn before it will use the despawn location, such as the end of a patrol route or a specific life span.

EX_n|R_m – Exit at Waypoint EX_n

The EX_n flag is used in exactly the same way as the EE flag except it takes over the exit function. The flag EX01 will cause the creature to despawn at a waypoint with the tag EE01. The creature will not spawn at that location. Used in conjunction with the EE flag, the creature will spawn at the EE_n flag and despawn at the EX_n flag. Using the optional R_m subflag will cause the creature to select a random waypoint with tags EX_n through EX_m for despawning. The creature must have a reason to despawn before it will use the despawn location.

To summarize:

- to get your creatures to spawn and despawn at another location use the EE flag,
- to get your creature to spawn at another location and despawn at a different location, use the EE flag with the EX flag.

Chapter 10 - Prepared for Death – CD

In this chapter we will:

- make the creature leave a lootable corpse rather than a loot bag,
- control how long the corpse will remain, and
- control what is left on the corpse.

CDn|Tm|Rq|D – Corpse Decay

Here is the flag that will get rid of all those fire beetle bellies! At last. The CD flag will leave an image of the corpse, and an invisible lootable object in the same location for the number of seconds specified in the required parameter. The required parameter n is an integer from 1 to 999. This is the standard method in the game for creating a lootable corpse. If you do not use the CDn flag the corpse will decay immediately and any inventory the creature had will be dropped in a loot bag. This can end up being very untidy. Using the CD flag and its subflags can clean up this mess.

The optional Tm subflag specifies what will remain on the lootable corpse depending on the value of the integer parameter m as follows:

- T0 = Inventory items only
- T1 = Inventory and equipped items
- T2 = Inventory items only, if the PC killed the creature
- T3 = Inventory and equipped items, if the PC killed the creature

The optional Rq subflag specifies the appearance of the remains left on the ground depending on the value of the integer parameter q as follows:

- R0 = Loot bag
- R1 = Body
- R2 = Bones
- R3 = Potion
- R4 = Pouch
- R5 = Scroll
- R6 = Treasure
- R7 = None (destroy loot on decay)

The Rq subflag determines the appearance of the remains and loot after the time set by the CD parameter. If there is loot remaining on the body after the CD parameter specified time period, the body and loot will transform into that specified by the Rq subflag parameter. If no R subflag is specified, the default value of R0 will be used. This will result in a loot bag left on the ground with any remaining loot in it. The loot bag will remain forever until it is emptied. The flag CD30R2 will leave a creature body for 30 seconds with all the creature's loot in the invisible corpse object. At the end of 30 seconds the body image and invisible corpse object will decay **if** there is no loot remaining. If loot remains, the body image will be replaced with the R2 specified image

of bones and will remain forever. It will only disappear if all the remaining loot is removed.

With previous versions of NESS, using the flag CD1 would cause all the creatures loot and equipped items to be place in a loot bag that would be decayed almost immediately. With the introduction of NESS 8.0, you must now add the R7 subflag to do this as the default is R0 when no R subflag is specified. The default, R0, will leave a loot bag where the creature's body fell. Use the flag *CDIR7* to ensure no loot or body remains. *CDIR7* will place all inventory in the invisible corpse object and decay it at the next cycle. The corpse image will immediately decay as well. A CD parameter of zero will not work, it will cause the CD flag to be ignored.

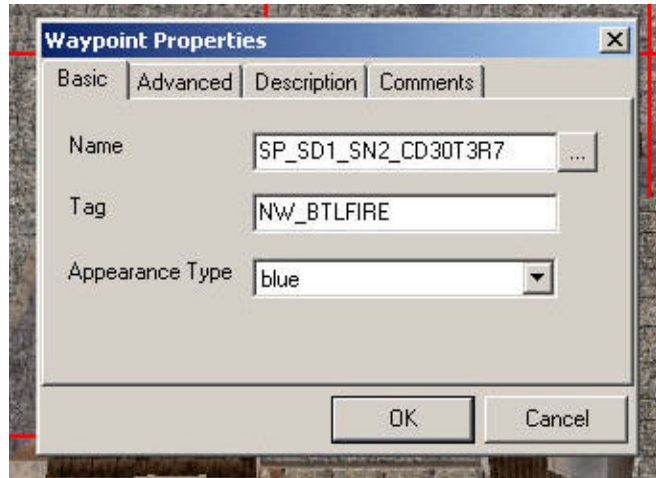


Figure 32 – Lootable beetle corpse.

The optional D subflag will cause the creature to drop any equipped weapons on the ground. Anything dropped on the ground will not be affected by what happens to the corpse image or invisible lootable corpse object.

Remember, only items indicated as droppable will end up in the invisible lootable corpse object or be dropped on the ground.

OK, ready to get rid of those beetle bellies? Go to the Town area of your module. Find the beetle waypoint and change the name field to *SP_SD1_SN2_CD30T3R7*. Remember that the invisible lootable corpse object will not be placed until the waypoint is processed at the next cycle. If you have set the SP parameter to a very high number the object may not get placed for several seconds.

Place the PC start point down the street to the east two tiles. Your waypoint properties should look like Figure 32. Click OK, save the module. Open the game for play.

The beetles will spawn one at a time at one-minute intervals. Kill one and the corpse object is created with the beetle belly inside. When the corpse decays it leaves nothing behind.

Now set the flags to *SP_SN2_CDIT3*. The corpse should decay at the next cycle, less than six seconds, without any refuse remaining.

The CD flag will drop inventory and equipped items that are marked droppable in the items properties dialog box. If the item is equipped and you don't want it to drop, remove it from the equipped slot into inventory, make sure the check box in the items properties

dialog box is NOT checked droppable, and then put it back into an equipped slot as shown in Figure 33. For some reason this process will not work if you change the items properties while in an equipped slot (thank you Sareena, ALFA Assistant Admin).

If you want to leave a clean corpse for some reason, use the creature's death script to get the corpse object with the code:

```
GetLocalObject(OBJECT_SELF,
"Corpse");
```

Then delete all the items in inventory.

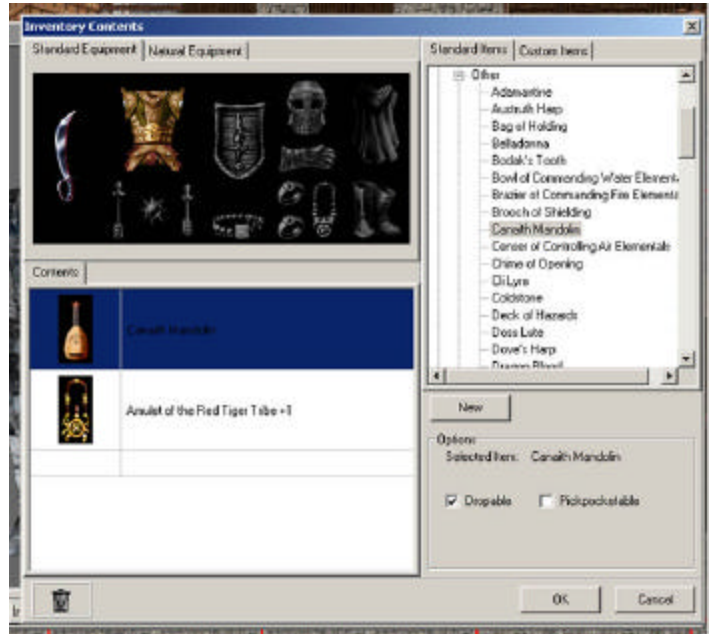


Figure 33 – Item checked as droppable.

To summarize:

- to get your spawned creatures to leave a lootable corpse, use the CD flag,
- use the CD parameter to specify the time the corpse will remain,
- to get the spawned creature or placeable to face a specific direction, use the SF flag,
- to get the spawned creature to sit, use the ST flag, and
- to put a probability that a spawn will occur on a spawn point, use the RS flag.

Chapter 11 - Spawn Unseen – PC, SU, and TR

In this chapter we will:

- learn how to make spawning occur without it being seen.

PC[n]R – Spawn Only If a PC is in the Area

Processing a hundred spawn points in each of a hundred areas might slow down your computer. Why not just skip the processing of the spawn points in the areas the PC is not in or expected to be in? That's what the PC flag can do for you. Include the PC flag alone and no spawning will be done when the PC is not in the area. Children will be spawned when the PC enters the area and immediately despawned when the PC leaves. To preserve the atmosphere of the experience, you probably do not want children spawned within sight of the PC at an area entry point. Some other method of controlling the timing of the spawn would need to be included. These methods will be discussed later.

The optional n parameter can be used with the PC flag to delay the despawning of the children when the PC leaves the area. This is useful when the PC is expected to return soon from the other area. This would be the case if the PC leaves the area to enter a small cave or house where he would not stay long. The optional n parameter is an integer greater than one that indicates the number of minutes to wait before despawning the children. A PC5 flag will spawn the children when the PC enters the area and despawn them when no PC has been in the area for five minutes.

Let's do an example in the *Forest* area. Deactivate all the other spawns in the area by putting an x before the SP in the waypoint name field. Place a waypoint exactly one tile due south of the entry point. Put the flag *SP3* in the waypoint name field and *nw_skeleton* in the tag field. Click OK. We are using the parameter three on the SP flag so we can kill the skeleton and exit the area before the next skeleton spawns. Open the *Town* area. Put the PC starting point near the transition to the *Forest* area. Save the module. Open the game in **Dungeon Master** (DM) mode for play.

When the game starts, open the *Chooser* and check the *Forest* area for creatures. There will be no creatures in the area and none will be spawned until you enter the area. Enter the area and you will see the skeleton spawn. Kill the skeleton and go back to the *Town* area. Check the *Chooser* for creatures in the *Forest* area. A skeleton will show up after a few seconds. Close the game. Once the creature has been spawned it will remain until despawned by some method.

By playing in DM mode we are able to see NPC activity in other areas of the module. While in the *Town* area, before entering the *Forest* area, there was no skeleton. When we entered the *Forest* area it spawned and, if killed, another spawned. When we returned to the *Town* area, the skeleton remained.

Now let's put the PC flag on the waypoint of the skeleton. Open the waypoint and change the name field to *SP3_PC*. Save the module. Open the game for play in DM mode.

When the game starts, open the *Chooser* and check the *Forest* area for creatures. You should see no creatures in the area. Enter the area and you will see the skeleton spawn after a few seconds (remember the waypoint cycle is eighteen seconds). Exit to the *Town* area. Check the *Chooser* again. The skeleton is not there. The skeleton has been despawned because there is no PC in the area, the skeleton was either despawned if there, or not spawned if not there.

Once more, let's change the PC flag on the waypoint of the skeleton. Open the waypoint and change the name field to *SP3_PCI*. Save the module. Open the game for play in DM mode.

Again when the game starts, open the *Chooser* and check the *Forest* area for creatures. You should see no creatures in the area. Enter the area and you will see the skeleton spawn after a few seconds. Exit to the *Town* area. Check the *Chooser* again. The skeleton is still there and will remain there for about a minute after you exit and will then be despawned. The skeleton's despawning has been delayed one minute after the last PC exited the area because of the parameter you set on the PC flag.

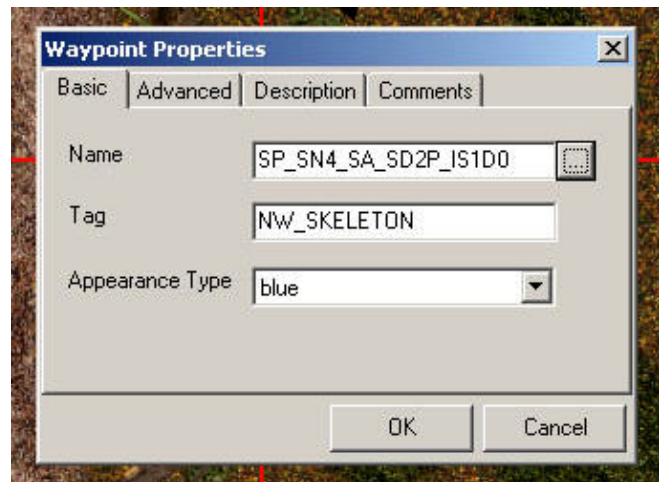


Figure 34 – Four skeletons for PC reset test.

Use the R subflag on the PC flag to reset the spawn point once all PCs have left the area. If you want an immediate reset once all PCs have left the area, use PCR.

So, what does a reset really do?

Go to the Forest area and create a spawn point two tiles in from the entrance on the road. Put the flags *SP_SN4_SA_SD2P_IS1D0* in the name field and *nw_skeleton* in the tag field. This will spawn four skeletons at once without delay and they will be respawned at two minute intervals if necessary regardless of when they are killed. Your waypoint properties will look like Figure 34. Place the PC start point two tiles to the east of the spawn point. Save the module, open it for play.

Enter the game in DM mode so we can see how the skeletons are behaving when outside the area. The four skeletons spawn immediately. Open the *Chooser* panel and check the list of creatures in the area. There will only be the DM and the four skeletons. Using your god like DM powers, kill two of the skeletons about a minute after entering the area. The *Chooser* now shows only two skeletons. Wait until two minutes after entering and the

two skeletons will be replaced. Kill two more skeletons. Move from the area into the *Town* area. About four minutes after you entered the *Forest* area check the *Chooser* for the creatures in the *Forest* area. You must do this by opening and closing the creature list. The *Chooser* does not update the creature list when it is open for creatures spawned and despawned by the NESS system. I don't know why, but that's the way it is. At about four minutes, two skeletons will be spawned to replace the ones you killed. The spawned creatures in the *Forest* area will remain there until the game is ended. The effect of not using the PC flag on a spawn point is that it behaves as if you never left the area. Close the game and open the Toolset.

Change the flags by adding the PC0 flag, your flag set will now be SP_SN4_SA_SD2P_IS1D0_PC0. Save the module and open it for play.

Again, open the game in DM mode. The four skeletons spawn immediately. Open the *Chooser* panel and check the list of creatures in the area. There will be the four skeletons. Kill two of the skeletons and exit the area. Check the list of creatures in the *Forest* area again. There are none. Because you added the PC0 flag to the spawn point the creatures were despawned as soon as the PC left the area. Enter the *Forest* area again. The two remaining skeletons are still there just as you had left them. You can expect the two missing skeletons you killed to be respawned at the appropriate interval. Close the game and open the Toolset.

Change the flags by changing the PC flag to PCR, your flag set will now be SP_SN4_SA_SD2P_IS1D0_PCR. Save the module, open it for play.

Again, open the game in DM mode. The four skeletons spawn immediately. Open the *Chooser* panel and check the list of creatures in the area. There will be the four skeletons. Kill two of the skeletons and exit the area. Check the list of creatures in the *Forest* area again. Nothing; the two skeletons have been despawned. Re-enter the area. All four of the skeletons have been respawned. The R subflag resets the spawn point to its initial condition as if it had not been spawned before.

Change the flags by changing the PC flag to PC1, your flag set will now be SP_SN4_SA_SD2P_IS1D0_PC1. Save the module and open it for play.

Again, open the game in DM mode. The four skeletons spawn immediately. Open the *Chooser* panel and check the list of creatures in the area. There will be the four skeletons. Kill two of the skeletons and exit the area within one minute of entering. Check the list of creatures in the *Forest* area again. There are only two skeletons. One minute after exiting the area the two remaining skeletons will be despawned because you have set the spawnpoint to despawn one minute after all PCs have departed the area. Enter the *Forest* area again. As was the case with the PC0 flag, the two remaining skeletons are still there and the two your PC killed will respawn at the time they would have had your PC not left the area.

Anyway, what did the R flag do for us? The R flag reset the spawn point to a condition as if the PC had not entered the area before.

SUn|[m] – Spawn Only if PCs are More Than n Meters Away

The SUn flag does not allow the spawn to occur if the PC is within n meters of the spawn waypoint, **not** the spawn location. It is most useful if the spawn is on a timer, such as SD, DO, NO, or HR and you do not want the PC to see the creatures spawn. The SUn flag will also help cut down on PC camping. A good number to use would be 30, this would be 30 meters or about three tiles. The PC usually can not see that far. This flag will also stop the spawning of creatures too close to a PC entrance, they will only spawn after the PC moves away.

The I[m] subflag is used to indicate the spawn location will be used instead of the location of the spawn waypoint that creates the spawn. If you have also used the SR or SL|R flag, then using the optional m parameter will cause the spawn to try m additional random locations at which to spawn while remaining n meters from the PC.

TRn|Dm – PC Proximity Trigger

Imagine you have a zillion creatures spawning in your area. Might slow down your computer a little bit, right? The TRn flag will help reduce the number of spawns active in your area. This flag tells the spawn point not to spawn until a PC is within n meters of the spawn point, where n is an integer greater than zero. The optional Dm subflag tells the spawn point to despawn the child creatures when the PC is not within m meters of the spawn point. If the D flag is not used the creatures will not be despawned unless some other method is being used to despawn them.

Open the dire rat waypoint and change the name field to *SP_SN3_SA3_TR10D20*, like that shown in Figure 35. Save the game, open for play.

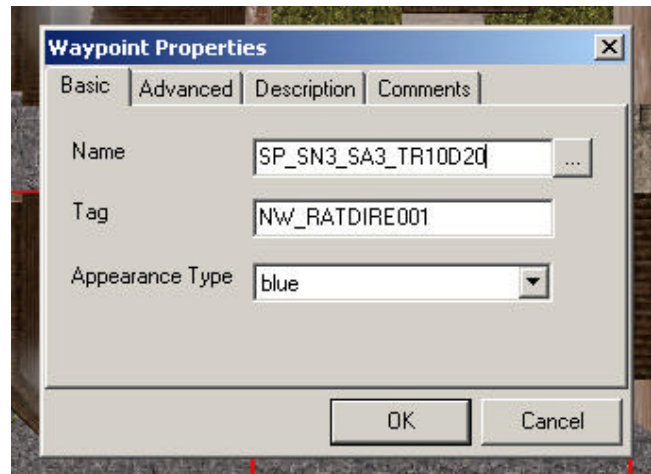


Figure 35 – Three dire rats with PC trigger.

What happened? Our PC is already two tiles, or 20 meters, away from the spawn point so there won't be any spawning occurring. As you move closer the three rats will eventually spawn and attack you because you are also within their perception range. If you turned and ran, they followed and did not despawn when you were much more than 20 meters away. Why? Because they were engaged in an activity, attacking you. They won't despawn if they're doing something. Try again, change the options to *SP_SN3_SA3_TR25D30*. This should spawn the creatures when the game starts and despawn when we retreat a small distance.

What happened? I have set up the demo area so that the rats are exactly two tiles or 20 meters away from my PC start position. This puts my PC within the spawn distance of the rats and outside their perception range. If I move my PC back 10 meters or one tile, I can see them despawn.

We don't actually want our game players to see the creatures spawn, so how do we use the TRnDn flag? Try this, deactivate the dire rat spawn by putting an x as the first character of the Name field, as shown in Figure 36. Move the fire beetle waypoint to the crossing of the red lines, one tile down from the now Xed out rats. Change the name field of the fire beetle waypoint to *SP_SN2_SA2_TR25D35* as shown in Figure 37.

By placing the fire beetle waypoint where we did, it is only visible by the PC if less than 20 meters away. By spawning at 25 meters we might give the beetles time to spawn if we are walking. Let's give it a try. Save the module. Open the game for play. Try approaching the beetles as soon as the game starts. Then clear out as fast as possible going back the way you came, and kill any beetles that follow. Go to the end of the block and wait a few seconds. Then run as fast as possible and approach the beetles from the south.

What happened? When you first rounded the corner the two beetles were waiting for you. If you killed them they respawned at the next cycle. If you walked all the way around the block and approached from the south the beetles would have despawned. When you ran down the street to the beetle spawn waypoint you probably just managed to see them finish spawning. This indicates the *TR25D35* needs to be changed to *TR35D45* for PC traffic coming fast down the street.

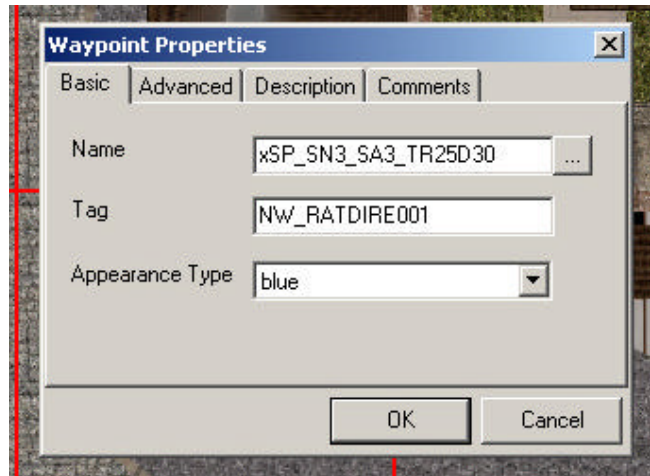


Figure 36 – x out the dire rats.

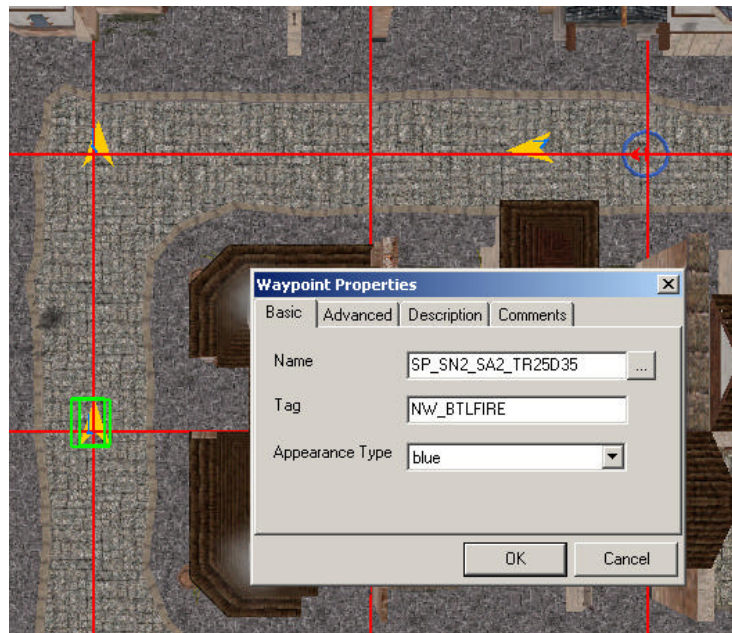


Figure 37 – Fire beetle flags and placement.

To summarize:

- the PC flag is used to control spawning and despawning based on the presence of the PC in the area,
- the SU flag is used to spawn children only if the PC is more than a specified distance away,
- the TR flag is used to control spawning and despawning based on how far the PC is from the spawn point.

Chapter 12 - Spawn Special Children – AL, FC, PT, CL, and HL

In this chapter we will:

- learn how to spawn with a specific alignment,
- learn how to spawn with a specific faction,
- learn how to spawn plot items,
- learn how to spawn children with specific life spans, and
- learn how to spawn self-healing children.

ALn|Sm – Spawn Creatures with a Specific Alignment

The ALn flag is used to adjust the alignment of the spawned creature. Before explaining what the AL flag does, a word about alignment. Alignment comes in two flavors, Good/Evil and Lawful/Chaotic. The alignment of the PC or creature on the Good/Evil and Lawful/Chaotic scale is measured with a number between 0 and 100. For the Good/Evil alignment, most Evil is assigned a value of 0 and most Good is assigned a value of 100. Likewise most Chaotic is assigned a value of 0 and most Lawful a value of 100. For both alignment scales neutral is 50. Let's talk about only the Good/Evil scale for the moment. When you want to make an adjustment to the Good/Evil alignment, you first select the direction in which you want to move alignment, towards Good, or towards Evil. See the table below for the parameter value associated with the direction you want to move the alignment. To move the alignment of the creature towards Good, use the AL3 flag and parameter, to move towards Evil, use AL4 flag and parameter. The S subflag and parameter m are used to specify the amount to move towards the selected end of the scale. AL3S10 will move the alignment of the spawned creature 10 points towards Good. If the template of the spawned creature started on the Good/Evil scale at 60, sort of Good, then the AL3S10 flag will result in a Good/Evil alignment of the spawned creature of 70. The parameter m of the S subflag must be an integer between 1 and 100, negative values are not allowed and a value of 0 does nothing. The Lawful/Chaotic alignment adjustments are done the same way using the AL1, more lawful, and AL2, more chaotic, flags and parameters.

n	alignment assigned
0	neutral
1	lawful
2	chaos
3	good
4	evil
5	all

What about the AL0 and AL5 flag and parameters work? The AL0 flag and parameter will move both the Good/Evil and Lawful/Chaotic alignments towards neutral, i.e., a

value of 50 by the amount specified by the S flag parameter. The change in alignment will at most take the value to 50 but not beyond it, above or below. As an example, let's say your spawned creature's template has a Good/Evil alignment of 25, middling Evil, and a Lawful/Chaotic alignment of 60, somewhat Lawful. The flag AL0S15 will result in moving the Good/Evil alignment 15 points towards neutral (50) or $25 + 15 = 40$. And moving the Lawful/Chaotic alignment 15 points towards neutral, or $60 - 15 = 45$, but not past 50, for a value of 50.

The AL5 flag will add the amount of the adjustment specified by the S subflag parameter to both the Good/Evil and Lawful/Chaotic alignments. Using the same starting creature as above, Good/Evil alignment of 25, and Lawful/Chaotic alignment of 60, a flag of AL5S10 will result in Good/Evil alignment of 35, and Lawful/Chaotic alignment of 70.

FCn – Spawn Creatures with a Specific Factions

The FCn flag is used to set the faction of the spawned creature. The parameter n is used to specify the faction according to the following table:

n	faction assigned
0	commoner
1	defender
2	merchant
3	hostile
4	custom

When using FC4, the spawn is suppose to take the faction of the nearest object. To use it you will want to place an object with the tag *SpawnFaction* and the desired faction next to the spawn point. Unfortunately, I have not been able to get this to work for custom factions.

PT – Spawn Children as Plot Objects

NESS sets the spawned object's plot flag.

CLn|Mm – Spawn Children with a Specific Life Span

Use the CLn flag to give the spawn a specific life span, the required parameter is an integer greater than one and is the life span in real world minutes. After the specified life span the creature or placeable will be despawned. Use the optional Mm subflag to give the spawned creatures or placeables a life span of between m and n minutes.

HL[n]E – Heal Children

The HL[n] flag will heal the spawned creatures n% each cycle. The default is 0% per cycle if n is not specified. HL1 is a special case and will heal the creature 100% per cycle.

The HL flag's parameter is a percentage and the result of the calculation is subject to truncation. If you have a creature with 8 HP and you specify HL10, 10% of 8 is less than one and no healing will ever be done.

The E subflag will remove all effects on the creature if the creature is at less than max HP.

To summarize:

- the AL flag will spawn a creature with a specific alignment,
- the FC flag will spawn a creature with a specific faction,
- the PT flag will tag the spawned creatures or placeables as plot items,
- the CL flag gives the spawned creatures and placeables a specific life span, and
- the HL flag will heal the spawned creatures a specified amount each cycle.

Chapter 13 - Spawning the Rich – RG

In this chapter we will:

- learn how to spawn creatures and containers with gold.

RGn|Mm|Cq – Spawn with Gold

Want your creatures to have a little spending money after pillaging the village? Add the RGn flag to supply the creature or container with a random amount of gold from 0 to n. Want a more narrow range of gold, add the Mm subflag to set a minimum amount of gold, RG100M50 will supply your pillaging orc with at least 50 gold to spend, but no more than 100. Want them to get an exact amount, RG100M100 will give them 100 gold exactly. Use the Cq subflag to give gold to only a certain percentage of the pillagers. Want only some of them to have gold? RG100M100C50 will give 100 gold to only 50% of the pillagers. Remember, the RG flag will only add to what is already in inventory, it will not replace inventory. You can use the RG flag in addition to the loot table flag LT discussed later.

To summarize:

- the RG flag is used to place gold on the creature or placeable container.

Chapter 14 - Spawning Placeables and Items – PL and IT

In this chapter we will:

- learn how to spawn placeables and items.

PL[n]|Tm|Pq – Spawning Placeables

Want to spawn a treasure chest full of gold along with those orcs? No problem. Placeables can be spawned using the flag PLn. The parameter determines the behavior of the placeable if it is a container. The parameter is an integer between 0 and 3.

PL0 = default behavior,

PL1 = despawn if the container is empty,

PL2 = refill if the container is empty, and

PL3 = refresh every q minutes, if no Pq subflag is used, the default is 60 minutes.

The PL flag **parameter** is only used for containers. If the placeable being spawned is not a container, like a campfire, it has no affect.

The Tm subflag determines the behavior of any traps set on the container. The parameter m determines the chance the trap will be disabled. The default is 100% chance of being disabled. The trap must already be a part of the placeables properties.

The Pq subflag sets the refresh period for the placeable container if the PL3 flag was used. If the placeable is not a container or PL0 through PL2 was used, it has no affect.

Remember that the tag field of the spawn waypoint must be the **resref** of the placeable. The tag of the placeable will only spawn vacant spaces! To find the resref of a standard NWN placeable, place the object in the area, open its properties, and get the resref off the *Advanced* tab. Usually the resref is the same as the tag, but lower case letters. If you try to edit the standard object and read the resref off the *Advanced* properties tab, it will not work. The resref shown in this case has been newly generated in the expectation you will be creating a custom object. Also, don't forget to include the PL flag in the flag set or you will spawn more vacant spaces.

Figure 38 is an example of the spawn waypoint properties for placing a treasure chest placeable. The RG flag will put 50 to 100 gold in it and it will have any other treasure the default NWN scripts gives it. It will not replenish or despawn unless your PC destroys it.

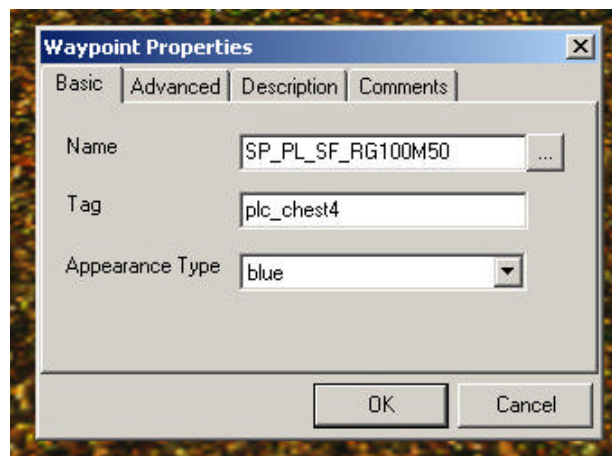


Figure 38 – Placing a treasure chest.

IT – Spawning Items

Put the IT flag in the flag set, put the **resref** of the item in the spawn waypoint tag field and get your item. It's that easy.

Figure 39 is an example of the spawn waypoint properties for placing a standard NWN magic sword.

To summarize:

- put the PL flag in the flag set, the resref of the placeable in the tag field of the spawn waypoint and get a placeable, or
- put the IT flag in the flag set, the resref of the item in the tag field of the spawn waypoint and get an item.

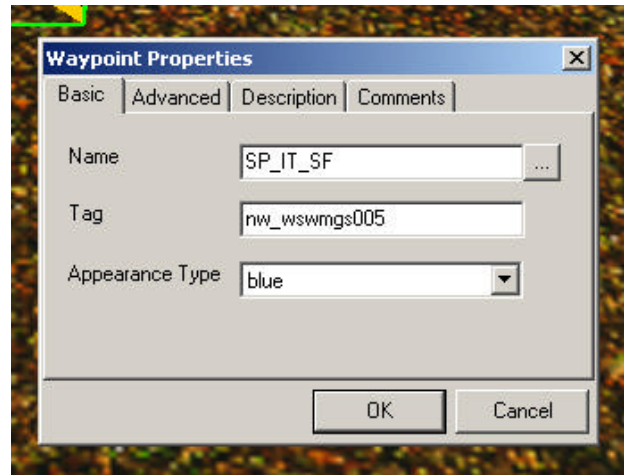


Figure 39 – Placing a magic sword item.

Chapter 15 - Deactivating the Spawned – DS and DI

In this chapter we will:

- learn how to deactivate a spawn,
- learn how to reactivate the deactivated spawn

DSn|Sm_DIq – Deactivate a Spawn

The DS n flag is used to deactivate the spawn point. The parameter n is used to specify the conditions that will cause deactivation. The flag DI q is used to provide additional deactivation information. It acts as a subflag for the DS flag and should be treated that way; it does nothing on its own. The DI flag is only needed when n is 3,4 or 5.

n	condition for deactivation
0	if all children are dead
1	if spawn number has been spawned
2	until all children are dead
3	after DI q children
4	after DI q minutes
5	after DI q cycles
6	when spawn count == spawn number

Once the spawn point has been deactivated, it can not be reactivated by the spawn point itself, i.e., any of the custom checks. Some other script must be employed to reactivate it. The subflag S m is one of the places you can put scripts to reactivate the deactivated spawn point. The parameter m specifies the entry in the spawn_sc_deactiv script to run when the spawn point is deactivated.

To summarize:

- put the DS flag in the flag set to deactivate the spawn point depending on the parameter settings,
- use the S subflag to run a script upon deactivation, and
- use the DI flag to provide additional deactivation options.

Chapter 16 - Spawning with Effect – AE, FX, and OE

In this chapter we will:

- learn how to spawn with effect,
- learn the difference between the effect flags.

AEn|Dm – Spawn with an Area Effect

The flag AEn generates an area effect at the time the creature spawns. The parameter n is an index supplied by you and is used in the spawn_cfg_fxae script to run the effect you have attached to the index number. The optional flag Dm sets the duration of the area effect in seconds with the parameter m. The default duration is five seconds if Dm is not specified. A D0 will make the effect permanent. The area effects are defined by constants with names that start with AOE, and these are the only effects the AE flag implements, 35 in total, listed in Appendix F.

Open the spawn_cfg_fxae script. The section of the script between the lines *Only Make Modifications Between These Lines* is shown in Figure 40. Two index number have already been defined, zero and one. The index zero returns no effect. Index one causes the effect AOE_PER_WEB to occur when the creature spawns.

```
//
// Only Make Modifications Between These Lines
// -----

// AreaEffect 00
// Dummy AreaEffect - Never Use
if (nSpawnAreaEffect == 0)
{
    return eAreaEffect;
}
//

// Web, only Visual
if (nSpawnAreaEffect == 1)
{
    eAreaEffect = EffectAreaOfEffect(AOE_PER_WEB, "*****", "*****", "*****");
}
//

// -----
// Only Make Modifications Between These Lines
//
```

Figure 40 – User defined section of spawn_cfg_fxae.

Let's try a simple example using the predefined effect, index one. Open the Toolset in the *Forest* area. Place the PC start point near the entrance on the north edge. Place a waypoint one tile south with the flags SP_SN5_AE1 and the tag *nw_rat001*. Save the module, open for play. What happened? Did rats spawn with the illuminated web effect? Good! It worked.

```

//
// Only Make Modifications Between These Lines
// -----

// All simple area effects
switch(nSpawnAreaEffect)
{
  case 0: return eAreaEffect; break;
  case 1: eAreaEffect=EffectAreaOfEffect(AOE_PER_WALLFIRE, "", "", ""); break;
  case 2: eAreaEffect=EffectAreaOfEffect(AOE_PER_FOGFIRE, "", "", ""); break;
  case 3: eAreaEffect=EffectAreaOfEffect(AOE_PER_FOGKILL, "", "", ""); break;
  case 4: eAreaEffect=EffectAreaOfEffect(AOE_PER_FOGMIND, "", "", ""); break;
  case 5: eAreaEffect=EffectAreaOfEffect(AOE_PER_WALLFIRE, "", "", ""); break;
  case 6: eAreaEffect=EffectAreaOfEffect(AOE_PER_WALLWIND, "", "", ""); break;
  case 7: eAreaEffect=EffectAreaOfEffect(AOE_PER_WALLBLADE, "", "", ""); break;
  case 8: eAreaEffect=EffectAreaOfEffect(AOE_PER_WEB, "", "", ""); break;
  case 9: eAreaEffect=EffectAreaOfEffect(AOE_PER_ENTANGLE, "", "", ""); break;
  //case 10: eAreaEffect=EffectAreaOfEffect(AOE_PER_CHAOS, "", "", ""); break;
  case 11: eAreaEffect=EffectAreaOfEffect(AOE_PER_DARKNESS, "", "", ""); break;
  case 12: eAreaEffect=EffectAreaOfEffect(AOE_MOB_CIRCEVIL, "", "", ""); break;
  case 13: eAreaEffect=EffectAreaOfEffect(AOE_MOB_CIRCGOOD, "", "", ""); break;
  case 14: eAreaEffect=EffectAreaOfEffect(AOE_MOB_CIRCCLAW, "", "", ""); break;
  case 15: eAreaEffect=EffectAreaOfEffect(AOE_MOB_CIRCCHAOS, "", "", ""); break;
  case 16: eAreaEffect=EffectAreaOfEffect(AOE_MOB_FEAR, "", "", ""); break;
  case 17: eAreaEffect=EffectAreaOfEffect(AOE_MOB_BLINDING, "", "", ""); break;
  case 18: eAreaEffect=EffectAreaOfEffect(AOE_MOB_UNEARTHLY, "", "", ""); break;
  case 19: eAreaEffect=EffectAreaOfEffect(AOE_MOB_MENACE, "", "", ""); break;
  //case 20: eAreaEffect=EffectAreaOfEffect(AOE_MOB_UNNATURALE, "", "", ""); break;
  case 21: eAreaEffect=EffectAreaOfEffect(AOE_MOB_STUN, "", "", ""); break;
  case 22: eAreaEffect=EffectAreaOfEffect(AOE_MOB_PROTECTION, "", "", ""); break;
  case 23: eAreaEffect=EffectAreaOfEffect(AOE_MOB_FIRE, "", "", ""); break;
  case 24: eAreaEffect=EffectAreaOfEffect(AOE_MOB_FROST, "", "", ""); break;
  case 25: eAreaEffect=EffectAreaOfEffect(AOE_MOB_ELECTRICAL, "", "", ""); break;
  case 26: eAreaEffect=EffectAreaOfEffect(AOE_PER_FOGHOUL, "", "", ""); break;
  case 27: eAreaEffect=EffectAreaOfEffect(AOE_MOB_TYRANT_FOG, "", "", ""); break;
  case 28: eAreaEffect=EffectAreaOfEffect(AOE_PER_STORM, "", "", ""); break;
  case 29: eAreaEffect=EffectAreaOfEffect(AOE_PER_INVIS_SPHERE, "", "", ""); break;
  case 30: eAreaEffect=EffectAreaOfEffect(AOE_MOB_SILENCE, "", "", ""); break;
  case 31: eAreaEffect=EffectAreaOfEffect(AOE_PER_DELAY_BLAST_FIREBALL, "", "",
  ""); break;
  case 32: eAreaEffect=EffectAreaOfEffect(AOE_PER_GREASE, "", "", ""); break;
  case 33: eAreaEffect=EffectAreaOfEffect(AOE_PER_CREEPING_DOOM, "", "", "");
  break;
  case 34: eAreaEffect=EffectAreaOfEffect(AOE_PER_EVARDS_BLACK_TENTACLES, "", "",
  ""); break;
  case 35: eAreaEffect=EffectAreaOfEffect(AOE_MOB_INVISIBILITY_PURGE, "", "",
  ""); break;
  case 36: eAreaEffect=EffectAreaOfEffect(AOE_MOB_DRAGON_FEAR, "", "", ""); break;
  case 99: eAreaEffect=EffectAreaOfEffect(AOE_PER_FOGACID, "", "", ""); break;
  default: return eAreaEffect; break;
}

// -----
// Only Make Modifications Between These Lines
//

```

Figure 41 – My user defined section of spawn_cfg_fxae.

I do not use the AE flag for anything other than applying these effects. Consequently, I have replaced all of the user defined portion of spawn_cfg_fxae with a simple switch statement using the values of the predefined constant effect names to implement them. This script is shown in Figure 41. For the AOE_PER_FOGACID I use the index 99. For all other values no effect is implemented. Of course other values may be used to implement other effects, such as a combination of effects. Remember to recompile the spawn_sample_hb script if you make any changes to the spawn_cfg_fxae script.

FXn|Dm – Spawn in with Effect

The FXn flag implements a visual effect when the creature spawns. The parameter n is an index supplied by you and is used in the spawn_cfg_fxsp script to run the effect you have attached to the index number. The optional Dm subflag implements a visual effect when the creature despawns, the parameters m is used in the same way as with the FX flag. The FXn flag implements those visual effects that have no user controlled duration, that is, those that are immediate in nature. These are all the visual effects defined by constants starting with VFX_IMP, VFX_COM, and VFX_FNF. Visual effects starting with VFX_DUR, VFX_ITM, and VFX_BEAM are all implemented with flag OE. All the visual effect constants are listed in Appendix E.

The user defined section of the spawn_cfg_fxsp script is shown in Figure 42. Two index number have already been defined, zero and 51. The index zero returns no effect. Index 51 causes the effect VFX_IMP_DISEASE_S to occur when the creature spawns.

```
//
// Only Make Modifications Between These Lines
// -----
//
// Effect 00
// Dummy Effect - Never Use
if (nEffectNumber == 0)
{
    return nEffect;
}
//
// Rotting Corpse Spawn/Despawn
if (nEffectNumber == 51)
{
    nEffect = VFX_IMP_DISEASE_S;
}
//
// -----
// Only Make Modifications Between These Lines
//
```

Figure 42 – User defined section of spawn_cfg_fxsp.

Let's try a simple example using the predefined effect, index 51. Open the Toolset in the *Forest* area. We'll use the waypoint from the example above. Change the waypoint flags to SP_SN5_FX51. Save the module; open for play. What happened? Rats spawned with the disease effect? Good, it worked.

As with the case of the spawn_cfg_fxae script file, I do not use the spawn_cfg_fxsp script format. All of the visual effect constants are numbers after all. So I just use the value of the constant to specify the effect. The script I use is shown in Figure 43. Of course, if you use a value to specify an effect that is not implemented by an FX effect, nothing will happen. For instance, value 17 is constant VFX_DUR_WEB, that is an effect not implemented by the FX flag.

Remember to recompile the `spawn_sample_hb` script if you make any changes to the `spawn_cfg_fxae` script. Let's try an example. Go back to our last example, the rats.

```
//
// Only Make Modifications Between These Lines
// -----

// Effect 00
// Dummy Effect - Never Use
if (nEffectNumber == 0)
{
    return nEffect;
}
else // all others
{
    nEffect = nEffectNumber;
}

// -----
// Only Make Modifications Between These Lines
//
```

Figure 43 – My `spawn_cfg_fxsp` user defined section.

Change the waypoint flags to `SP_RW_CLI_FX57` and the tag to `dwarfmerc002`. This will spawn a dwarf guard with a life of one minute with the `VFX_IMP_DOOM` effect. Save the module, open the game for play; dwarf spawns with black cloud over head. Good. Go back to the toolset. Change the flags to `SP_RW_CLI_FX24D235`. Save the module, open the game for play. The dwarf spawns in with an implosion effect, `VFX_FNF_IMPLOSION`, and despawns after one minute with the large red chunks effect, `VFX_COM_CHUNK_RED_LARGE`.

OEn|Dm – Spawn with Object Effect

The OEn flag implements the remainder of the VFX defined effects whose constant names start with `VFX_DUR`, `VFX_ITM`, and `VFX_BEAM`. All of these effects have user defined durations. All the visual effect constants are listed in Appendix E.

The parameter `n` on the OE flag is an index supplied by you and is used in the `spawn_cfg_fxobj` script to run the effect you have attached to the index number. The optional `Dm` subflag sets the duration of the object effect in seconds with the parameter `m`. The default duration is five seconds if `Dm` is not specified. A `D0` will make the effect permanent.

The user defined section of the `spawn_cfg_fxobj` script is similar to that for `spawn_cfg_fxsp` shown in Figure 42. Again, two index number have already been defined, zero and one. The index zero returns no effect. Index one causes the effect `VFX_DUR_BARD_SONG` to occur when the creature spawns.

As with the case of the `spawn_cfg_fxsp` script file, I do not use the `spawn_cfg_fxobj` script format. I just use the value of the constant to specify the effect. The script I use is the same as shown in Figure 43. Of course, if you use a value to specify an effect that is

not implemented by the OE flag, nothing will happen. For instance, value 24 is constant VFX_FNF_IMPLOSION - that is an effect not implemented by the OE flag.

To summarize:

- use the AE flag to implement an area visual effect when the creature is spawned,
- use the AE subflag D to implement an area visual effect when the creature despawns,
- use the FX flag to implement an immediate object visual effect when the creature spawns,
- use the FX subflag D to implement an immediate object visual effect when the creature despawns,
- use the OE flag to implement an object visual effect when the creature spawns and has a user definable duration.

Chapter 17 - Spawning a Merchant – SM

In this chapter we will:

- learn how to spawn a merchant.

SM – Spawn a Merchant

Spawn a merchant object by including the SM flag in the flag set. The tag field of the spawn point must be the resref of the standard NWN merchant or the custom merchant. If you have created a custom merchant to spawn, make sure the tag and the resref are not the same. If the tag and the resref are the same then the spawned merchant object waypoint and the spawn waypoint that created it will both have the same tag and are located in the same place. When the merchant NPC tries to open the merchant object the script used to open it uses the function `GetObjectByTag()`. This action might get the spawned merchant object or the spawn waypoint that created it. In addition to making sure the resref and tag are different, use the script in Figure 44 to open the merchant. This script will be placed in the *Actions Taken* for the line of conversation that starts the merchant. It looks for the matching tag that is a merchant object. This script is especially

```
#include "nw_i0_plot"
void main()
{
    //Either open the store with that tag or let the user know that no store exists.
    int nCount = 1;
    string sStoreTag = "trusty_bow_and_throw"; // Set this to what you want
    object oStore = GetNearestObjectByTag(sStoreTag);
    while(GetIsObjectValid(oStore))
    {
        if(GetObjectType(oStore) == OBJECT_TYPE_STORE)
        {
            {
                gplotAppraiseOpenStore(oStore, GetPCSpeaker());
                return;
            }
        }
        oStore = GetNearestObjectByTag(sStoreTag, OBJECT_SELF, ++nCount);
    }
    ActionSpeakStringByStrRef(53090, TALKVOLUME_TALK);
}
```

Figure 44 – Actions Taken script to open merchant.

important if you are spawning a standard NWN merchant because you can not make the tag and resref different, they are always the same and unchangeable.

Let's create a bow and arrow merchant to spawn.

- Take any of the standard merchants and make a copy.
- Remove all the items in inventory. You might want to save this empty custom merchant for later use.
- Place only bows and arrows in the now empty merchant inventory.

- Give the custom merchant a name like Trusty Bow & Throw.
- Give it a tag of *trusty_bow_and_throw*.
- Give it a resref of *trusty_bow* as show in Figure 45 and Figure 46.
- Create the custom merchant by clicking OK in the Merchant Properties dialog box.
- Set a spawn waypoint with the name field *SP_SM_PC2*.
- Put the resref of the custom merchant in the tag of the spawn waypoint, *trusty_bow* as shown in Figure 47.

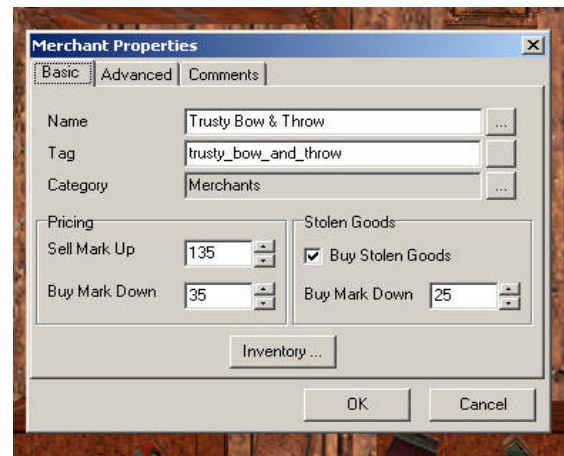


Figure 45 – Merchant Properties Basic.

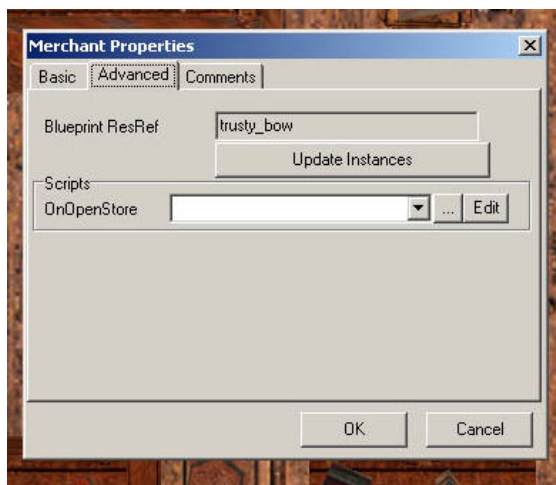


Figure 46 – Merchant Properties Advanced.

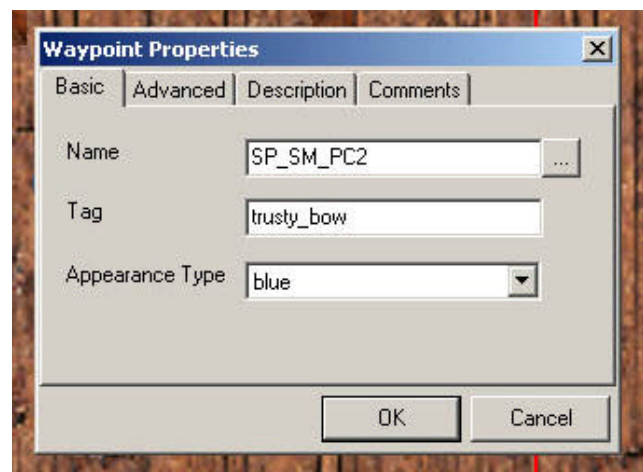


Figure 47 – Merchant spawn waypoint.

Now the spawn waypoint will spawn the merchant with the resref of *trusty_bow*. To use it, have the NPC merchant execute the script shown in Figure 44 at the point in the conversation where you want the merchant to open. Be sure to replace the *oStore* variable with the tag of your merchant.

Each time the merchant is spawned, the inventory will be reset to the original inventory of the merchant object. Why would you want to spawn a merchant? Because the inventory is reset to the original inventory each time it is spawned. This is most useful in a persistent world module. The merchant will destroy anything it has bought from the PC and replenish anything it has sold to the PC. If you have the merchant object spawn each time the PC enters the merchant's shop, the merchant's inventory will always be the same.

The only other flags that might be of use with the SM flag are the DOD, NOD and HR flags so that the merchant is only open during certain hours. If you replace the line,

```
ActionSpeakStringByStrRef(53090, TALKVOLUME_TALK);
```

with,

```
ActionSpeakString("Sorry, we're closed", TALKVOLUME_TALK);
```

then the merchant NPC will let the PC know the store is closed.

To summarize:

- use the SM flag to spawn a merchant,
- you can spawn your own custom merchants or the standard NWN merchants,
- make sure the tag and resref of your custom merchants are not the same, and
- use the script in Figure 44 to open spawned merchants.

Chapter 18 - Loot Tables – LT

In this chapter we will:

- learn how give your spawn loot.

LTn|Am|Bq|Cr – Spawn Loot on Creatures Using Loot Tables

The LT flag is used to place loot on your creature or in your placeable container. The loot placed on the creature or in the container is defined in the loot table in the script file `spawn_cfg_loot`. The user defined portion of `spawn_cfg_loot` is shown in Figure 48. Not all of the user defined portion is shown, only the first two lines of script for loot tables greater than 499 are shown.

User defined loot table scripts

To use the loot tables, add the LTn flag in the flag set for your spawned creature, such as SP_LT0. The number zero is the parameter attached to the LT flag in this case. In the loot table the parameter following the LT flag will be assigned to the variable *nLootTable*. This variable is used to select the script that will be used to give the loot to the creature. Figure 48 shows, the script of the loot table for *nLootTable* = 0 creates an item with the resref of *magic sword* and puts it on the creature. Don't expect this to work unless you create a custom item with the resref of *magic sword*. When you imported NESS, you did not get a custom item named *magic sword*. Anyway, that's how you use the loot table in its simplest form. The subflags A, B, and C are not used with the user defined loot tables.

DanieleB's merchant based loot tables

NESS 8.0 has another method of giving loot scripted by DanieleB. DanieleB's method uses a merchant object created by you. To use DanieleB's merchant based loot tables:

1. Use the Merchant Wizard to create a merchant. When the Merchant Wizard asks for a name for the merchant, give it a name in the format *LOOT_n*, where n is an integer greater than 499. For instance *LOOT_555*. This will make the name and tag field of the custom merchant *LOOT_555*. It is only important that the tag field have the name *LOOT_555*.
2. Once the custom merchant has been created, right click on it and edit it. Add or subtract items from the merchant's inventory to create the pool from which loot will be selected. You can have one item or many, custom items or standard items.
3. Place an instance of the merchant in any area in your module. It does not have to be placed in the area you will be using it. Or, spawn it as described in the previous chapter.

4. Add the flag *LTn* to your flag set where *n* is the number you used to name your merchant, in this example, *LT555*.

You can create other *LOOT_n* merchants for different types of loot, such as high value treasure or cursed treasure. Then use the different *LOOT_n* merchants on different spawned creatures. For example liches may give cursed treasure.

```
//
// Only Make Modifications Between These Lines
// -----

// Table 00
if (nLootTable == 0)
{
    // 50% Chance
    if (d100(1) > 50)
    {
        // Created Custom Item with ResRef of magicword
        sTemplate = "magicword";
        nStack = 1;
    }
    oItem = CreateItemOnObject(sTemplate, oSpawned, nStack);
}
//

// Random Gold and *Nothing* Else
else if (nLootTable == 1)
{
    CleanInventory(oSpawned);
    // Add our Items: Gold for Example
    nStack = Random(50) + 1;
    oItem = CreateItemOnObject("nw_it_gold001", oSpawned, nStack);
}
//

// Merchant-based loot - from DanieleB NESS scripts
else if( nLootTable >= 500 )
{
    object oStore = OBJECT_INVALID;
    oStore = GetObjectByTag( "LOOT_" + IntToString(nLootTable));
    .
    .
    .
    .
}
// -----
// Only Make Modifications Between These Lines
//
```

Figure 48 – User defined loot tables.

The subflags *A*, *B*, and *C* are used only with the merchant based loot tables. The *A* subflag is the probability of one item being placed in the creature/container inventory, the *B* subflag is the probability of two items being placed in the creature/container inventory, and the *C* subflag is the probability of three items being placed in the creature/container inventory.

If the general formula for the flag is $LtnAmBqCr$, then:

- the probability of three items in inventory of creature/container = r% (parameter of C),
- the probability of two items in inventory of creature/container = r% + q% (parameter of B+C),
- the probability of one item in inventory of creature/container = r%+q%+m% (parameter of A+B+C), and
- all items will be randomly selected from merchant named LOOT_n.

LT501A30B20C10 would put three items in inventory 10% of the time, two items 30% of the time (20%+10%) and one 60% (30%+20%+10%), and none 40% of the time from merchant LOOT_501.

LT502A0B0C100 would put three items into the creature/container inventory 100% of the time. The parameters on the A and B subflags are irrelevant in this case.

LT503A100B50C50 would put three items on the creature/container 50% of the time and two 100% of the time; the parameter on the A subflag is irrelevant.

If no A, B, or C subflag is used, the defaults are A50B15C5. LT555 would put three items in inventory 5% of the time, two items 20% of the time (5%+15%) and one 70% (50%+15%+5%), and none 30% of the time from merchant LOOT_555.

Cyrus_1469 DMG loot tables

Cyrus has written scripts to make loot giving easy and enjoyable. The scripts randomly select from all standard magic items in the game by category, gold, and a set of ten gems. Cyrus has created twenty loot table entries, one through twenty, that follow the DMG guide as closely as possible. Loot table entry one is shown in Figure 49.

```
//
// Only Make Modifications Between These Lines
// -----
.
.
.
// Calling Loot Table 1
if (nLootTable == 1)
{
    GenerateGold(oSpawned, 80, 10, 100);
    GenerateGems(oSpawned, 0, 1, 4);
    GenerateMagic(oSpawned, 0, 3);
}
.
.
.
// -----
// Only Make Modifications Between These Lines
//
```

Figure 49 – Sample Cyrus loot table entry.

To put gold or gems on your creature, use the functions:

```
GenerateGold(p1, p2, p3, p4);
GenerateGems(p1, p2, p3, p4);
```

The parameters of the functions GenerateGold and GenerateGems are:

p1 = always oSpawned, creature to get loot, do not change,
 p2 = percent chance creature will get any gold or gems,
 p3 = minimum stack size, and
 p4 = maximum stack size.

The line GenerateGold(oSpawned, 40, 1000, 6000) will give 1,000 to 6,000 gold 40% of the time. The line GenerateGems(oSpawned, 60, 10, 40) will give a stack of 10 to 40 gems 60% of the time.

```
//
// Only Make Modifications Between These Lines
// -----
.
.
// Calling Loot Table 99
if (nLootTable == 99)
{
  GenerateArmor(oSpawned);
  GenerateWeapon(oSpawned);
  int i = d8()+d3();
  switch (i)
  {
    case 1: GeneratePotions(oSpawned); break;
    case 2: GenerateScrolls(oSpawned); break;
    case 3: GenerateRings(oSpawned); break;
    case 4: GenerateRods(oSpawned); break;
    case 5: GenerateStaffs(oSpawned); break;
    case 6: GenerateWands(oSpawned); break;
    case 7: GenerateCloaks(oSpawned); break;
    case 8: GenerateBoots(oSpawned); break;
    case 9: GenerateBelts(oSpawned); break;
    case 10: GenerateBags(oSpawned); break;
    case 11: GenerateNecklace(oSpawned); break;
  }
}
.
.
// -----
// Only Make Modifications Between These Lines
//
```

Figure 50 – Custom Cyrus loot table entry.

Cyrus has tried to match the DMG loot guide lines as best he can. Nevertheless, there are some inconsistencies between Neverwinter Nights and the DMG loot guides. Neverwinter Nights will not stack more than ten gems, so even if you ask for a stack of 20 to 30 gems in the script, you will only get a stack of ten.

The function GenerateMagic will provide one of 13 types of magic items, such as armor, wands, weapons, etc. To put magic items on your creature, use the function:

```
GenerateMagic(p1, p2, p3);
```

The parameters of the function GenerateMagic are:

p1 = always oSpawned, creature to get loot, do not change,
 p2 = percent chance creature will get any item,
 p3 = number of NWN standard magic items.

The line GenerateMagic(oSpawned, 60, 3) will give three magic items 60% of the time.

Want to specify a particular item, such as armor, scrolls, or rings? Use one of the 13 functions that give specific items. If you want to provide one random piece of standard NWN magic armor, add

```
GenerateArmor(oSpawned);
```

to your loot table entry. It is just as easy for swords, helmets, etc. Figure 50 is an example of a custom loot table entry using Cyrus' functions. It will give the creature or container one magic armor item, one magic weapon, and one other magic item, gem, bag, or potion.

Cyrus' loot tables are available on the Internet at nwwvault.ign.com/Files/scripts/data/1034838714431.shtml.

- Download the *rar* file and unzip it. It will contain two erf files, LootTable.erf and Items.erf. LootTable.erf contains the script files and Cyrus' spawn_cfg_loot replacement script. Items.erf contains the custom items used in his loot tables.
- Import the Items.erf files.
- **Before importing the LootTables.erf file save your existing spawn_cfg_loot script to another name, such as spawn_old_loot. If you do not save the old spawn_cfg_loot file you will lose DanieleB's merchant based loot table script and any other custom scripts you have already entered in to the spawn_cfg_loot script file.**
- Import the LootTables.erf file.
- After you have imported both erf files, open the spawn_cfg_loot script and spawn_old_loot script files in the script editor.
- Copy and paste any custom scripts you had in the old file as well as DanieleB's merchant loot table script into the new spawn_cfg_loot.

Appendix G – Cyrus' & DanieleB's Loot Tables, is a complete replacement for the spawn_cfg_loot script that includes both Cyrus' and DanieleB's loot tables. You can use this file if you don't have any custom scripts in your spawn_old_loot script. You will still need Cyrus' include files in your script collection.

Things to remember:

- The loot tables are called when the creature/item is spawned, not when it dies.
- You can also use the RG flag along with the LT flag.

To summarize:

- the LT flag is used to put loot on the creature or container when it is spawned,
- you can create your own custom loot table entries,
- NESS 8.0 comes with DanieleB's merchant based loot tables, and
- Cyrus 1469 loot table entries are an easy addition to the existing entries.

Chapter 19 - Spawning a Group – SG

In this chapter we will:

- learn how to use the SG flag so creatures will be selected from the group table,
- learn how to specify where in the group table creatures will be selected, and
- learn how to define the parameters of creature selection from the group table.

SG – Spawn a Group

Up until now all the creatures spawned at one waypoint have all been the same creature, the one specified in the tag field of the waypoint properties. If you wanted a different type of creature, you had to put down another waypoint.

Group tables allow you to spawn any of several different creatures with just one waypoint depending on the method you choose to select the creatures in the group table.

First, how do we get the waypoint to use the group table to select creatures rather than specifying the exact creature in the tag field of the waypoint? The SG flag is placed in the name field of the waypoint, along with any other flags you want to define the actions of the creatures.

Second, in the tag field of the waypoint properties, where you had previously placed the tag or resref of a creature, you put a unique template name that will indicate the place in the group table to select the creatures. The unique identifier is made up by you.

Third, you make a place in the group table, using your unique template, that selects the creature tag or resref that will be spawned.

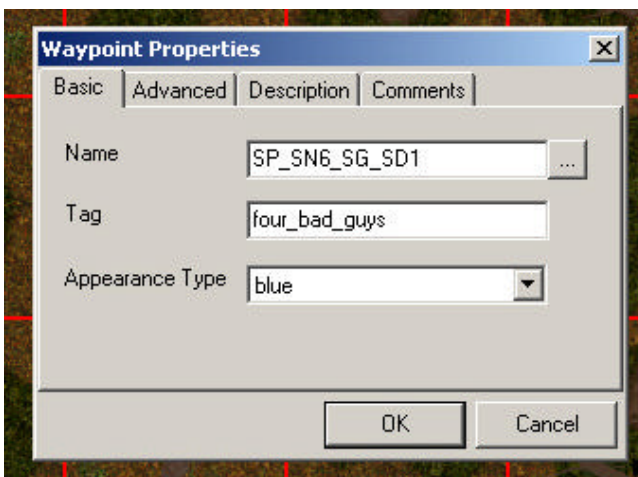


Figure 51 – four_bad_guys group.

I'm sure a good example is needed here. Go to the three campfires located in the southwest corner of the Forest area. Delete any waypoints that create creatures. We want six creatures spawned that will be randomly selected from the following four types of standard creatures, the nw_skeleton, the nw_orca, and the two goblins nw_goblina and nw_goblinb. Create a waypoint next to the campfires and put the following flags in the name field, *SP_SG_SD1_SN6*. We need a name to call this selection of goblins, skeletons, and orcs, *four_bad_guys* will do. Place the name *four_bad_guys* in the tag field of the waypoint properties. Your waypoint should look like Figure 51.

Now let's move to the `spawn_cfg_group` file. Open the file for editing. Each time a creature is to be spawned by a waypoint with an SG flag, a call is made to the function `SpawnGroup` in the `spawn_cfg_group` script. The function imports the name of our custom group, `four_bad_guys`, that we put in the waypoint tag field, in the variable `sTemplate`. The function returns a single creature tag or resref that will be spawned in the variable `sRetTemplate`. In the group table, look for the section delimited by the comments *Only Make Modifications Between These Lines*. This is the section of the function where we can add the script necessary to assign a value to the return variable `sRetTemplate`. Add the script shown in Figure 52.

What does this script do? First, it takes the incoming variable `sTemplate` and tests it to determine if it matches the name `template` of the group we entered in the tag field of the waypoint. Remember, we can have as many of these group name templates as we want for other spawn points. This script will be called six times in this case because we are creating six creatures with this waypoint. Each time it is called we want it to select one of the four creature tags to return. This is done with a simple switch selection using the `d4(1)` function for input. Each time this spawn point needs a creature tag, it calls this function, rolls the dice, assigns a tag, and returns that tag to the spawn generator.

```
//
// Only Make Modifications Between These Lines
// -----
.
.
.
// four_bad_guys
if (sTemplate == "four_bad_guys")
{
    switch(d4(1))
    {
        case 1:
            sRetTemplate = "nw_skeleton";
            break;
        case 2:
            sRetTemplate = "nw_orca";
            break;
        case 3:
            sRetTemplate = "nw_goblina";
            break;
        case 4:
            sRetTemplate = "nw_goblinb";
            break;
    }
}
//
.
.
.
// -----
// Only Make Modifications Between These Lines
//
```

Figure 52 – User defined group `four_bad_guys`.

Let's give it a try. Save the `spawn_cfg_group` file, recompile the `spawn_sample_hb` script, and save the module. Note that we had to do a compile. This is necessary each

time you modify a configuration file. Open the game for play. Approach the camp, but stay out of perception range.

```

//
// Only Make Modifications Between These Lines
// -----
.
.
.
// scaled encounter
if (sTemplate == "scaled_encounter")
{
    // Initialize Variables
    int nTotalPCs;
    int nTotalPCLevel;
    int nAveragePCLevel;
    object oArea = GetArea(OBJECT_SELF);
    // Cycle through PCs in Area
    object oPC = GetFirstObjectInArea(oArea);
    while (oPC != OBJECT_INVALID)
    {
        if (GetIsPC(oPC) == TRUE)
        {
            nTotalPCs++;
            nTotalPCLevel = nTotalPCLevel + GetHitDice(oPC);
        }
        oPC = GetNextObjectInArea(oArea);
    }
    if (nTotalPCs > 0)
    {
        nAveragePCLevel = nTotalPCLevel / nTotalPCs;
    }
    else
    {
        nAveragePCLevel = 1;
    }
    // Select a Creature to Spawn
    switch (nAveragePCLevel)
    {
        // Spawn Something with CR 1
        case 1:
            sRetTemplate = "nw_skelton";
            break;
        //
        // Spawn Something with CR 5
        case 5:
            sRetTemplate = "nw_skelmage";
            break;
    }
}
.
.
.
// -----
// Only Make Modifications Between These Lines
//

```

Figure 53 – Scaled encounter group.

What happened? Six creatures, representing a random selection from the four bad guys, are standing around the campfires. Kill them all and wait for the respawn. The next set of creatures should be different than the first but still made up of the four bad guys.

So, maybe you want a scaled encounter, that is, spawn a creature or creatures depending on the hit points of all the PCs in the area. The script in Figure 53 would do this. It sums the total hit dice for all the PCs in the area, determines an average, and sets sRetTemplate

depending on the average hit dice determined. You will have to include more values, and your own creatures, in the switch statement of course. You could also use a random selection method to select which creature to return for each case.

Different Loot for Different Creatures in a Group

In your SpawnGroup, where you specify which creature to spawn for that pass, you would just add the line:

```
SetLocalInt(oSpawn, "f_LootTable", 501);
```

where 501 is the number of the loot table entry to use, as shown in Figure 54.

```
//
// Only Make Modifications Between These Lines
// -----
.
.
.
// different loot
if (sTemplate == "variable_loot_group")
{
    switch(d4(1))
    {
        case 1:
            sRetTemplate = "nw_skeleton";
            SetLocalInt(oSpawn, "f_LootTable", 501);
            break;
        case 2:
            sRetTemplate = "nw_orca";
            SetLocalInt(oSpawn, "f_LootTable", 502);
            break;
        case 3:
            sRetTemplate = "nw_goblina";
            SetLocalInt(oSpawn, "f_LootTable", 503);
            break;
        case 4:
            sRetTemplate = "nw_goblinb";
            SetLocalInt(oSpawn, "f_LootTable", 504);
            break;
    }
}
//
.
.
// -----
// Only Make Modifications Between These Lines
//
```

Figure 54 – Variable loot.

Specific Creatures with Specific Loot in a Group

Now you want a specific number of specific creatures with specific loot for each in your group. Use a counter integer located on the spawn waypoint to keep track of the number of creatures already spawned. The value of the counter is used to determine the next

creature and its loot. You will need to have an SN flag in your flag set to set the number of creatures to spawn. Figure 55 is the script in the spawn_cfg_group table for this.

```
//
// Only Make Modifications Between These Lines
// -----
.
.
.
// Spawn one of each
if (sTemplate == "one_of_each")
{
    int i_one_of_each = GetLocalInt(OBJECT_SELF, "one_of_each_counter");
    switch (i_one_of_each)
    {
        case 0: sRetTemplate = "nw_skeleton";
                SetLocalInt(oSpawn, "f_LootTable", 501);
                break;
        case 1: sRetTemplate = "nw_skelwarr01";
                SetLocalInt(oSpawn, "f_LootTable", 502);
                break;
        case 2: sRetTemplate = "nw_skelwarr02";
                SetLocalInt(oSpawn, "f_LootTable", 503);
                break;
        case 3: sRetTemplate = "nw_skelmage";
                SetLocalInt(oSpawn, "f_LootTable", 504);
                break;
        case 4: sRetTemplate = "nw_skelpriest";
                SetLocalInt(oSpawn, "f_LootTable", 505);
                break;
    }
    i_one_of_each++;
    if (i_one_of_each >= 5) i_one_of_each = 0;
    SetLocalInt(OBJECT_SELF, "one_of_each_counter", i_one_of_each);
    return sRetTemplate;
}
.
.
.
// -----
// Only Make Modifications Between These Lines
//
```

Figure 55 – Specific creatures with specific loot.

To summarize:

- the SG flag is used to spawn a group of placeables and creatures,
- any number of placeables and creatures may be in the group, and
- you must recompile the script containing the Spawn() function (spawn_sample_hb) to realize the changes made to the spawn_cfg_group file.

Chapter 20 - Spawning a Camp – CM

In this chapter we will:

- learn how to make camp.

CM|Dn – Spawn a Camp With Creatures and Placeables

A camp is a template for a set of creatures and placeables all spawned at once. All the spawned placeables and creatures are the children of an invisible object that is the child of the spawn point. The definition of the camp is placed in the `spawn_cfg_camp` script file between the lines *Place Custom Camps Here*. The `spawn_cfg_camp` script file contains two predefined camps, a goblin camp and a demon camp that you can use as further examples. To create a camp add the flag CM to the SP flag in the waypoint name field, and the waypoint tag field is the name of the custom camp. The optional Dn flag despawns the placeables n seconds after all the creatures despawn or die.

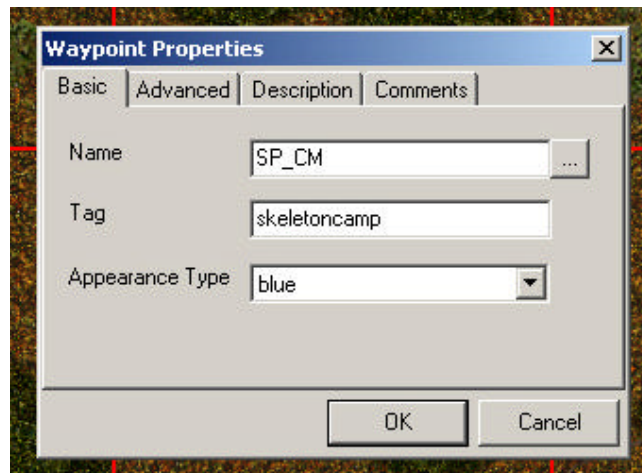


Figure 56 – Skeleton camp waypoint.

Let's make a camp named *skeletoncamp*. Open the *Forest* area of your module, go to the southeast corner of the area and place a waypoint in the tile second from the bottom edge and second from the right edge. The waypoint name field will be *SP_CM* and the waypoint tag will be *skeletoncamp*. Figure 56 is an example of our camp waypoint.

Our custom camp will consist of:

- one skeleton mage,
- one skeleton warrior,
- three skeletons,
- one fire pit, and
- one treasure chest.

By placing the CM flag on the waypoint, the spawn system looks to the `spawn_cfg_camp` script for the creatures and placeables to spawn. The variable `sCamp` in the `SpawnCamp` function is set to the name of the camp, in our case, *skeletoncamp*.

You define a camp in the `spawn_cfg_camp` script file by adding the following lines in the area of the script designated for camp definitions.

1. Insert the if Statement to test for our camp name.

```
if (sCamp == "skeletoncamp") {
```

2. Set the number of creatures that will be spawned with the line:

```
SetLocalInt(oCamp, "CampNumC", 5);
```

where five (5) is the number of creatures in the camp. You will define the creatures later. You may have any number of camp creatures, or none.

3. Set the number of placeables for the camp with the line:

```
SetLocalInt(oCamp, "CampNumP", 2);
```

where two (2) is the number of placeables in the camp. You will define the placeables later. Again, you may have any number of camp placeables, or none.

4. Define the radius within which all the creatures will be placed with:

```
SetLocalFloat(oCamp, "CampRadius", 5.0);
```

where five (5.0) sets the radius for five meters. Creatures will be placed randomly within the circle.

5. Now you need to define the creatures and the flags that will go with each creature just as if you were creating a waypoint. The only difference is that you will be putting the creature tags and flags into variables in the oCamp object. You will define the first creature, C0, with the following two lines:

```
SetLocalString(oCamp, "CampC0", "nw_skelmage");
SetLocalString(oCamp, "CampC0_Flags", "SP");
```

The variable CampC0 holds the creature tag and the variable CampC0_Flags holds the flags for the first creature. Since you've specified five creatures, you will need to repeat these lines for CampC1, CampC2, CampC3, and CampC4 and their corresponding flags, CampC1_Flags, CampC2_Flags, CampC3_Flags, and CampC4_Flags.

```
SetLocalString(oCamp, "CampC1", "nw_skelwarr01");
SetLocalString(oCamp, "CampC1_Flags", "SP");
SetLocalString(oCamp, "CampC2", "nw_skeleton");
SetLocalString(oCamp, "CampC2_Flags", "SP");
SetLocalString(oCamp, "CampC3", "nw_skeleton");
SetLocalString(oCamp, "CampC3_Flags", "SP");
SetLocalString(oCamp, "CampC4", "nw_skeleton");
SetLocalString(oCamp, "CampC4_Flags", "SP");
```

6. You will define each placeable in a similar manner. The only difference is you will be putting the placeables resref into the variable in the oCamp object.

```
SetLocalString(oCamp, "CampP0", "plc_campfrwspit");
SetLocalString(oCamp, "CampP0_Flags", "SP");
SetLocalString(oCamp, "CampP1", "plc_campfrwspit");
SetLocalString(oCamp, "CampP1_Flags", "SP");
```

You have now defined the camp placeables and creatures. This is all that is required for a camp.

7. Optionally, you may define the center of the camp with one of the previously defined creatures or placeables. This is done by assigning a string to the variable CampCenter located on the oCamp object. In this instance use the string C0 through C4 or P0 through P1 to define the center of the camp to be creature CampC0 through CampC4 or placeable CampP0 or CampP1 respectively. We will use placeable CampP0.

```
SetLocalString(oCamp, "CampCenter", "P0");
```

If you do not designate a CampCenter, the CampP0 object will be the camp center by default.

8. Optionally, you may define a script to run if one specific creature dies or a placeable is destroyed. The CampTrigger variable designates which creature or placeable to monitor and the CampTriggerScript is an index to the script to run when the object dies or is destroyed. The script that will be executed is placed in the spawn_sc_cmpt trig script file. Trigger index 0 has been predefined to destroy the entire camp. If we set creature C0 as the CampTrigger, the skeleton mage, and the script index to 0, then the camp, all creatures and placeables, will be destroyed if the skeleton mage is killed. Let's do that.

```
SetLocalString(oCamp, "CampTrigger", "C0");
SetLocalInt(oCamp, "CampTriggerScript", 0); } //close if parentheses
```

That's it, your basic skeleton camp. The finished script should look like that in Figure 57. Let's give it a try. Save the spawn_cfg_camp script, open the spawn_sample_hb and compile it. Place the PC starting point near the entrance to the area along the north edge. Save the module and open the game for play.

What happens? Go to the southeast corner and see the skeletons standing around. Kill the skeleton mage and see the entire camp despawn and a new one spawn immediately. Close the game, return to the toolkit.

There are a few flags that can be added to each creature's *CampP1_Flags* flag settings. They are RW, SF, SG, LT, CD, and PL.

```

//
// Place Custom Camps Here
// -----

// Example Camp
// Campfire, chest, skeleton mage, skeleton warrior, 3 skeletons
if (sCamp == "skeletoncamp")
{
    // Set Number of Creatures
    SetLocalInt(oCamp, "CampNumC", 5);
    // Set Number of Placeables
    SetLocalInt(oCamp, "CampNumP", 2);
    // Set Radius of Camp
    SetLocalFloat(oCamp, "CampRadius", 5.0);

    // Set Creature 0 and Spawn Flags
    SetLocalString(oCamp, "CampC0", "nw_skelmage");
    SetLocalString(oCamp, "CampC0_Flags", "SP");

    // Set Creature 1 and Spawn Flags
    SetLocalString(oCamp, "CampC1", "nw_skelwarr01");
    SetLocalString(oCamp, "CampC1_Flags", "SP");

    // Set Creature 2 and Spawn Flags
    SetLocalString(oCamp, "CampC2", "nw_skeleton");
    SetLocalString(oCamp, "CampC2_Flags", "SP");

    // Set Creature 3 and Spawn Flags
    SetLocalString(oCamp, "CampC3", "nw_skeleton");
    SetLocalString(oCamp, "CampC3_Flags", "SP");

    // Set Creature 4 and Spawn Flags
    SetLocalString(oCamp, "CampC4", "nw_skeleton");
    SetLocalString(oCamp, "CampC4_Flags", "SP");

    // Set Placeable 0 and Spawn Flags
    // First Placeable always Spawns at Center of Camp
    // If CampCenter Is Not Set
    SetLocalString(oCamp, "CampP0", "plc_campfrwspit");
    SetLocalString(oCamp, "CampP0_Flags", "SP");

    // Set Placeable 1 and Spawn Flags
    SetLocalString(oCamp, "CampP1", "chest01");
    SetLocalString(oCamp, "CampP1_Flags", "SP");

    // Set Placeable 0 to be Camp Center
    SetLocalString(oCamp, "CampCenter", "P0");

    // Set Creature 0 to be Trigger
    // Script 00 : Kill him and the Camp Despawns
    SetLocalString(oCamp, "CampTrigger", "C0");
    SetLocalInt(oCamp, "CampTriggerScript", 0);
}
.
.
.
// -----
// Only Make Modifications Between These Lines
//

```

Figure 57 – User defined camps.

OK, let's see if we can make this a little bit more interesting by using some of those flags we already know about. Add the optional D60 flag to the CM flag on the camp spawn waypoint. This will delay the despawning of the camp for sixty seconds so you will have time to loot the treasure from the treasure chest. Add the NOD flag to the existing flags in the waypoint name field so that the camp will spawn only during the night and despawn

at dawn. This way the camp will only exist at night. Add an SD12 so the camp will not respawn for twelve minutes after despawning. With this setting, and assuming you still have the module setting of one minute per game hour, the camp will not spawn again during the night. The camp waypoint flags are now *SP_CM_NOD_SD12*.

To each of the individual creature flags add *RW_CD60T3*. Now the flag for each creature will be *SP_RW_CD60T3*. The creatures will randomly walk and leave a corpse for 60 seconds. To the treasure chest placeable add the *RG100M50* flag so it will have some gold in it. The flag for the chest placeable will now be *SP_RG100M50*. Save the *spawn_cfg_camp* script, open the *spawn_sample_hb* and compile it. Save the module and open the game for play.

Now your camp will not spawn until hour 18 and will despawn at hour 6 if you don't kill the creatures. If you kill the creatures and/or the skeleton mage, the camp will not respawn until the next night. The treasure chest will have some gold in it each night as well.

This is just the basic way to build a camp. Since camps are built using strings set as local variables, they can easily be built dynamically by the system. Any logic you need can be placed in the *spawn_cfg_camp* file, very similar to the logic arrangements used in the group tables. The possibilities are endless.

One that I will mention is to define several camps in the *spawn_cfg_camp* script file and select randomly from them using the following the script shown in Figure 58.

```

if (sCamp == "randomcamp") {
    switch (d2(1)) {
        case 1: // Camp 1 goblins
            SetLocalInt(oCamp, "CampNumP", 1;
            SetLocalInt(oCamp, "CampNumC", 1;
            SetLocalFloat(oCamp, "CampRadius", 5.0);
            SetLocalString(oCamp, "CampP0", "plc_campfrwspit");
            SetLocalString(oCamp, "CampP0_Flags", "SP_SF");
            SetLocalString(oCamp, "CampC0", "nw_gobchiefb");
            SetLocalString(oCamp, "CampC0_Flags", "SP_RW");
            break;
        case 2 // Camp 2 skeletons
            SetLocalInt(oCamp, "CampNumP", 1;
            SetLocalInt(oCamp, "CampNumC", 1;
            SetLocalFloat(oCamp, "CampRadius", 5.0);
            SetLocalString(oCamp, "CampP0", "plc_chest001");
            SetLocalString(oCamp, "CampP0_Flags", "SP_SF");
            SetLocalString(oCamp, "CampC0", "nw_skeleton");
            SetLocalString(oCamp, "CampC0_Flags", "SP_RW");
            break;
    }
}

```

Figure 58 – Random camp.

To summarize:

- the CM flag is used to spawn a camp,
- the user picked name of the camp is placed in the tag field of the spawn waypoint, and
- the camp is defined in the *spawn_cfg_camp* script.

Chapter 21 - Flag Tables – FT

In this chapter we will:

- learn what the flag table is, and
- learn the methods of using the flag table.

FT[n] – Use Flags from Flag Table

Custom scripts are generally called in one of two ways; by a number attached to the flag, or by a name placed in the spawn waypoint name. The flag table is unique in that it can use both methods.

The flag table can be used in two ways. First, the flag FT can be placed in the name field of the spawn waypoint and a variable specified in the tag field. The variable would then be used as an entry in the flag table to replace the old flags with new flags and replace the variable with a creature resref. Second, a parameter can be used with the FT flag to specify a new set of flags and creatures in the flag table or additional flags that could be appended to the flags in the waypoint name field. Let's look at each one of these different ways of using the flag table.

Using the FT flag with a variable in the Tag field of the waypoint and replacing the original flag set and/or creatures

The first method of using the flag table is to put the FT flag in the flag set without any parameter following it, or the parameter 0, i.e., FT or FT0. When you choose this option, the tag field of the spawn waypoint is assumed to be a variable name that will be used in the flag table similarly to the name of a group in the group table. Let's look at the flag table for an example. Figure 59 is a listing of the user defined section of the flag table. When FT or FT0 is used in the flag set, the variable *nFlagTableNumber* in the flag table is equal to zero and the script following the *if (nFlagTableNumber == 0)* is executed. Following this *if* statement is another that tests the variable *sSpawnTag*. The variable *sSpawnTag* contains the creature template of the tag field in the original spawn waypoint. Depending on the original contents of the tag field you can replace the flag set and/or the creatures to be spawned. Assign the new flag set string to the variable *sFlags* and the new creature template string to the variable *sTemplate*. Using the first example in the flag table, you would get the following transformation:

	Original spawn		New spawn
Field	waypoint properties	Converts to	waypoint properties
Name:	SP_FT	→	SP_SN02_SA_RW
Tag:	myspawns	→	nw_dog

When FT is used without a parameter, the string in the tag field of the spawn waypoint is used as a variable to define a new set of flags and creatures.

```

//
// Only Make Modifications Between These Lines
// -----
// Sample Complex Replacement
// Using FT without FT00 will
// Default to nFlagTableNumber 0
if (nFlagTableNumber == 0)
{
    // Old Method of using SpawnTag
    if (sSpawnTag == "myspawns")
    {
        sFlags = "SP_SN02_SA_RW";
        sTemplate = "nw_dog";
    }

    if (sSpawnTag == "undead")
    {
        sFlags = "SP_SNO4";
        sTemplate = "nw_zombie01";
    }
}
//

// Sample Simple Replacement Flag
// Completely Replaces Flags
// On Spawnpoints with FT01
if (nFlagTableNumber == 1)
{
    sFlags = "SP_SN04_RW_DOD";
    sTemplate = "nw_dog";
}
//

// Sample Template Flags
// These Flags Get Added
// To Spawnpoints with FT02
if (nFlagTableNumber == 2)
{
    sFlags = "_RW_PC05R";
}
//
// -----
// Only Make Modifications Between These Lines
//

```

Figure 59 – User defined section of flag table.

Using the FT flag with a parameter and replace the original tag set and/or creatures

The second method of using the flag table is to put the FTn flag in the flag set where n is an integer parameter greater than zero. When you choose this option, the variable *nFlagTableNumber* in the flag table is equal to n and the script following the *if (nFlagTableNumber == n)* is executed. Following this if statement you can replace the flag set and/or the creature template with script. Assign the new flag set string to the variable *sFlags* and the new creature template string to the variable *sTemplate*. Using the third example in the flag table, you would get the following transformation:

	Original spawn		New spawn
Field	waypoint properties	Converts to	waypoint properties
Name:	SP_FT1	➔	SP_SN02_RW_DOD
Tag:	???	➔	nw_dog

Both of the methods allow you to amend the flag set rather than replacing it entirely. If the string assigned to *sFlags* starts with the characters SP then the flag set will be completely replaced with the new string. If the string assigned to *sFlags* does not start with the characters SP the string will be amended to the original flag set without the FT flag. For example, to replace the original flag set with a new flag set, assign the new flag set string to the variable *sFlags*, such as *sFlags* = "SP_SN04_RW_DOD". To amend the flag set, assign the additional flag string, starting with the underscore character, to the variable *sFlags*, such as *sFlags* = "_RW_PC05R". Using the last example in the flag table, you would get the following transformation:

	Original spawn		New spawn
Field	waypoint properties	Converts to	waypoint properties
Name:	SP_FT2	➔	SP_RW_PC05R
Tag:	NW_DOG	➔	nw_dog

Lastly, remember to recompile the `spawn_sample_hb` script when you change any `cfg` script file.

To summarize:

- use the FT flag without a parameter to use the tag field as a variable to set flags and creature templates in the flag table, or
- use the FT flag with a parameter greater than zero to use the parameter as a variable to set flags and creature templates in the flag table.

Chapter 22 - Spawn Script – SS

In this chapter we will:

- learn how to run a script when the creature is spawned.

SSn|Dm – Run an OnSpawn Script When Children are Spawned or Despawned

Custom scripts are generally called in one of two ways; by a number attached to the flag, or by a name placed in the spawn waypoint name. Browsing through the files will give you an idea of which is which, but for the most part, groups, camps, and parts of the flag tables are called by name and the rest are called by script numbers attached to the flag.

Spawn scripts are called by attaching a number to the SS flag. Add the SS n flag to the flag set to have the spawned creature run script number n defined in the spawn_sc_spawn script file. The script will run once when the creature is spawned.

The Dm subflag has the creature run script m on despawning.

Spawn Invisible

Want to spawn an invisible creature, or at least invisible until it attacks its first unlucky victim? Create an entry in the spawn_sc_spawn script file, say number 90. Attach to it the following command:

```
DelayCommand(1.0, ActionCastSpellAtObject(SPELL_INVISIBILITY,  
OBJECT_SELF, METAMAGIC_ANY, TRUE));
```

The DelayCommand is to make sure the ActionCastSpellAtObject is performed after all other actions that might cause it to become visible again. Add the flag SS90 to the flag set of the creature.

Chapter 23 - Heartbeat Script – HB

In this chapter we will:

- learn how to run a script at each game heartbeat for a spawned creature.

HBn – Creatures Will Run a Script on Heartbeat

Custom scripts are generally called in one of two ways, by a number attached to the flag, or by a name placed in the spawn waypoint name. Browsing through the files will give you an idea of which is which, but for the most part, groups, camps, and parts of the flag tables are called by name and the rest are called by script numbers attached to the flag.

Heartbeat scripts are called by attaching a number to the HB flag. Add the HBn flag to the flag set to have the spawned creature run script number n defined in the *spawn_sc_hbeat* script file. The script will run on every heartbeat, so use the HB flag sparingly if you have a slow computer.

Chapter 24 - Custom Code – CP and CC

In this chapter we will:

- learn how to check spawn.

CPn|Rm – Check PC Custom Script to Determine if Spawn Proceeds

Custom scripts are generally called in one of two ways, by a number attached to the flag, or by a name placed in the spawn waypoint name. Browsing through the files will give you an idea of which is which, but for the most part, groups, camps, and parts of the flag tables are called by name and the rest are called by script numbers attached to the flag.

Check PC scripts are called by attaching a number to the CP flag. Add the CPn flag to the flag set to have the spawned creature run script number *n* defined in the *spawn_chk_pcs* script file. The script will run on every processing cycle of the spawnpoint to determine if the spawn proceeds. When using the CP flag, the spawn will not occur unless the variable `nProcessSpawn` is set to TRUE somewhere in the *spawn_chk_pcs* script.

CCn – Check Custom Script to Determine if Spawn Proceeds

Custom scripts are generally called in one of two ways, by a number attached to the flag, or by a name placed in the spawn waypoint name. Browsing through the files will give you an idea of which is which, but for the most part, groups, camps, and parts of the flag tables are called by name and the rest are called by script numbers attached to the flag.

Custom check scripts are called by attaching a number to the CC flag. Add the CCn flag to the flag set to have the spawned creature run script number *n* defined in the *spawn_chk_custom* script file. The script will run on every processing cycle of the spawnpoint to determine if the spawn proceeds. When using the CC flag, the spawn will not occur unless the variable `nProcessSpawn` is set to TRUE somewhere in the *spawn_chk_custom* script.

If you are doing checks against PCs, it's probably best to place them in the *spawn_chk_pcs* since the majority of the time, *spawn_chk_custom* won't have any PCs to check against. Also, the *spawn_chk_pcs* already cycles through all the PCs in the area.

A simple example will illustrate its use. A spawn waypoint can be triggered by a trigger using the CC flag. Set a spawn waypoint with the flags SP_CC2 and tag `nw_skeleton`. Create a trigger with the tag *trigger2*. Add the script shown in Figure 60 to the OnEnter event. Put the script shown in Figure 61 in the *spawn_chk_custom* script file.

The trigger will set the local variable *TriggerTrip* to true when a PC enters the trigger with the script in Figure 60. The next processing of the spawn point will spawn the creature because the custom check script now will return *nProcessSpawn = TRUE*. The local variable *TriggerTrip* will be reset to false three seconds after the creature spawns.

```
void main()
{
    if (GetIsPC(GetEnteringObject()))
        SetLocalInt(OBJECT_SELF, "TriggerTrip", TRUE);
}
```

Figure 60 – Trigger OnEnter event.

This way only one creature will be spawned when the trigger is triggered and then reset for the next entering PC.

```
if (nSpawnCheckCustom == 2)
{
    int nTriggerTrip = GetLocalInt(GetObjectByTag("trigger2"), "TriggerTrip");
    if (nTriggerTrip)
    {
        nProcessSpawn = TRUE;
        DelayCommand(3.0, SetLocalInt(GetObjectByTag("trigger2"), "TriggerTrip",
FALSE));
    }
}
```

Figure 61 – Custom check script.

To summarize:

- use the CP flag to create a custom PC check to spawn creatures spawns only under certain conditions set by the script in *spawn_chk_pcs*,
- use the CC flag to create spawns only under certain conditions set by the script in *spawn_chk_custom*.

Chapter 25 - Death Scripts – DT

In this chapter we will:

- learn how to run a script when the creature dies.

DTn – Run Script on Death of Creature

Custom scripts are generally called in one of two ways; by a number attached to the flag, or by a name placed in the spawn waypoint name. Browsing through the files will give you an idea of which is which, but for the most part, groups, camps, and parts of the flag tables are called by name and the rest are called by script numbers attached to the flag.

Death scripts are called by attaching a number to the DT flag. Add the DTn flag to the flag set to have the spawned creature run script number *n* defined in the *spawn_sc_death* script file. The script will run once when the creature dies.

Chapter 26 - Proper ID and Flagging the Flag – ID and CF

In this chapter we will:

- learn how to use the ID flag to locate a specific spawn waypoint, and
- learn how to use the CF flag.

IDn – Give a Spawn Waypoint an ID number

The IDn flag actually does nothing. Use it to easily identify the spawn point for later use. Let's try an example of why and how to use it. We'll create a situation where the PC enters an area and is immediately set upon by five skeletons who rush at the PC from distant random locations. This will only happen once when entering the area. To do this, put the spawn point as near the entry point as possible, use the *Forest* area and put the PC start point in the *Town* area near the transition into the *Forest* area. Use the flags *SP_SN5_SA_IS1D0_DS1_SS98_ID98_SL05R01* in the spawn waypoint name field and put *nw_skeleton* in the tag field. This waypoint will spawn five skeletons (SN5) all at once (SA), immediately (IS1D0), deactivate the spawn point after the five skeletons have spawned

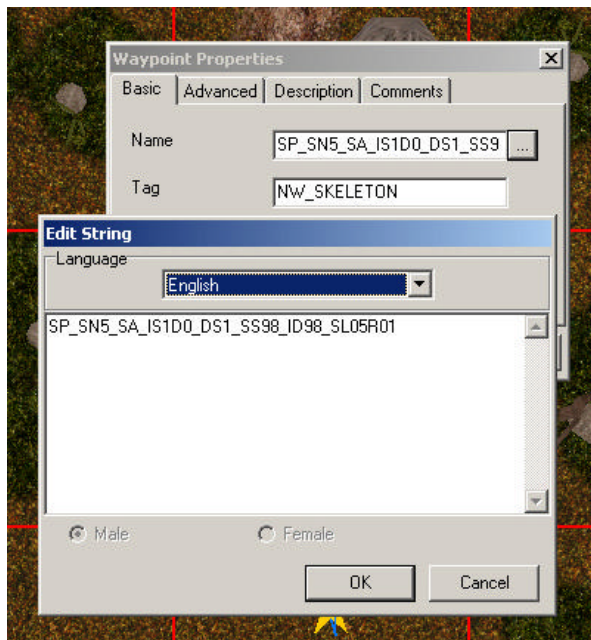


Figure 62 – Five skeletons closing on PC.

```
//
// Only Make Modifications Between These Lines
// -----
.
.
.
if (sSpawnScript == 98)
{
    ActionMoveToObject(GetSpawnByID(98), TRUE, 1.0);
    return;
}
.
.
.
// -----
// Only Make Modifications Between These Lines
//
```

Figure 63 – spawn_cc_spawn run to spawn point script.

(DS1), run spawn script 98 (SS98), has ID98 (ID98), and will spawn the five skeletons at five random waypoint locations (SL05R01). The spawn waypoint is shown in Figure 62. The five random waypoint spawn locations will have the waypoint tags SL01 through

SL05 (See the discussion about the SL flag). Place these five waypoints three tiles from the spawn waypoint in a semicircle. To get the skeletons to run to the spawn waypoint from the actual spawn locations a script will be necessary. The script will be placed in the `spawn_cc_spawn` script file with the index 98. The script is shown in Figure 63 and uses the built in NESS function `GetSpawnByID()`. The script is very simple. It gets the spawn point object with the ID 98 and then has the creature run to the object. After adding the new script, save the `spawn_cc_spawn` script, open the `spawn_sample_hb` script and compile it, save the module, open the game for play.

Take the transition to the Forest area. As soon as you arrive the five skeletons spawn at one of the five waypoints which are beyond your PC's perception. They immediately begin running towards the spawn point that you placed near the entry to the Forest area. Kill them if you wish, return to the Toolset. The skeletons will not respawn because the spawn point was deactivated, so this is a one trick spawn point. If you want to reactivate the spawn point you will have to use the `NESS_ActivateSpawnByID` function in a script.

The three NESS functions that use the ID number are:

```
object NESS_GetSpawnByID(int nSpawnID, object oArea);
void    NESS_ActivateSpawnByID(int nSpawnID, object oArea);
void    NESS_DeactivateSpawnByID(int nSpawnID, object oArea);
```

If you know the area and ID, you can work with the spawn. These Activate and Deactivate functions only set a flag indicating that the spawn should be activated or deactivated but do not do anything else. The spawn will occur at the next processing of the spawn point. If the spawn point is not expected to be processed for some time, due to an SD or SPx setting, an immediate processing can be forced with the use of the function:

```
NESS_ForceProcess(object oSpawn);
```

made after the call to the activate or deactivate functions.

CF – Store Flag Information

Place the CF flag in the flag set and everything after the CF flag will be stored in a local string variable named *CustomFlag* on the spawned child. Make sure the CF flag is the last flag in the set as any flags after the CF flag will have no affect on the spawn. The spawn waypoint flags `SP_SN4_SA_CFGGeorge` would store the string *George* in the local string variable *CustomFlag*.

So, an example of how to use this would be nice, you say. Suppose you have one custom character and you want to give that character different conversation files for different spawn points. Change the custom character's *OnSpawn* script to trigger the *OnDialog* event by uncommenting the line:

```
SetSpawnInCondition(NW_FLAG_ON_DIALOGUE_EVENT);
```


found at about line 224 of the default OnSpawn script. Be sure to save the changed OnSpawn script to a new name. This will allow you to use the OnDialog event handler in the UseDefined script.

Add the script shown in Figure 64 to the creatures UserDefined script file. Now you can spawn several different creatures from the one custom creature with different conversations. Use the following spawn waypoint flags, *SP_CFconversation_01*, to get the spawned creature to use *conversation_01*. Change what follows the CF flag on the next spawn waypoint to get the same creature with a different conversation. Now, if I

```

.
.
.
case 1004:// on Dialog Event
{
    string sConversation = GetLocalString(OBJECT_SELF, "CustomFlag");
    object oPC = GetNearestPC();
    ActionStartConversation(oPC, sConversation);
}
break;
.
.
.
```

Figure 64 – On Dialog Event script in the UserDefined script file.

could just find a way to change the spawned creatures name I would only have to make one custom commoner to spawn many unique commoners with different conversations and different names. But, alas, I have not heard of a way to do that.

To summarize:

- use the ID flag to give the spawned creature a easy handle for later use,
- use the CF flag for passing unique information to the spawned creature.

Chapter 27 – Functions for External Control

In this chapter we will:

- learn about some addition functions you can use in your own scripts.

There are several functions and variables in the spawn system that can be used in your scripts to control the spawn points and the creatures. Remember, if you use any of the spawn system functions or variables you must add *spawn_functions* as an include with the line

```
#include "spawn_functions"
```

in your script file before the “void main” line.

The function

```
void NESS_ForceProcess(object oSpawn)
```

will force spawn waypoint oSpawn to be processed on the next heartbeat regardless of SPxx value. This will be used often in conjunction with the following functions and variables.

As mentioned in the preceding chapter, there are three functions that use the spawn point ID.

```
object NESS_GetSpawnByID(int nSpawnID, object oArea)
```

This function returns the spawn waypoint in area oArea with ID nSpawnID.

```
void NESS_ActivateSpawnByID(int nSpawnID, object oArea)
```

This function activates the spawn waypoint in area oArea with ID nSpawnID.

```
void NESS_DeactivateSpawnByID(int nSpawnID, object oArea)
```

This function deactivates the spawn waypoint in area oArea with ID nSpawnID. These activate and deactivate functions only set a flag telling the spawn waypoint to activate or deactivate itself. Of course, if the spawn waypoint is already deactivated it can not activate itself so some other means must be found. This is where the `NESS_ForceProcess(object oSpawn)` function comes in handy. To reactivate an inactive spawn waypoint with ID equal 17, use the following lines in your script:

```
NESS_ActivateSpawnByID(17, GetArea(OBJECT_SELF));
NESS_ForceProcess(NESS_GetSpawnByID(17, GetArea(OBJECT_SELF)));
```

This will set the spawn waypoint to active and force the spawn waypoint to process at the next game heartbeat.

Two other functions do the same thing without the aid of the ID flag.

```
void    NESS_ActivateSpawn(object oSpawn)
void    NESS_DeactivateSpawn(object oSpawn)
```

You will need the spawn waypoint object to use them. If you have not used an ID flag on the spawn waypoint you will have to use some other method of getting the spawn waypoint object.

The function

```
NESS_ProcessDeadCreature(OBJECT_SELF);
```

is used to process the dead creatures spawned by the spawn system. Why would I need this you may ask? You'd need it in the situation where you have used the spawn system to spawn a specific number of creatures and then deactivated the spawn waypoint. When the creatures are killed the spawn waypoint must be active to create a corpse image, corpse object to contain the loot, to transfer the loot to the corpse object, and finally to decay the corpse at the appropriate time. If the spawn waypoint is inactive, none of this will occur even if you use the CD flag on the spawn waypoint. To make sure it does occur, place the above function on the spawned creatures *OnDeath* event. You will have to create and spawn custom creatures to do this.

The following two functions

```
void    NESS_TrackModuleSpawns(int nFlag=TRUE)
int     NESS_IsModuleSpawnTracking()
```

enable and disable module tracking and return the setting of module tracking. Module tracking sends messages to any DM regarding the number of creatures in the area.

You can use the function

```
SetLocalInt(OBJECT_SELF, "ForceDespawn", TRUE);
```

instead of the DestroyObject command and the creature will take advantage of all special despawning circumstances such as scripts and exit locations. You would still need to wrap it into an ActionDoCommand statement and insert it into the action queue as follows:

```
ActionDoCommand(SetLocalInt(oNPC, "ForceDespawn", TRUE));
```

It has been my experience that the creature will not move to an exit point when this method is used. Therefore I insert code to make the NPC move to the desired exit location before using this method.

I find this command particularly useful when I want an NPC to be spawned only under specific conditions, interact with the PC, and then despawn, never to be seen again. Use the custom code flag CC to cause the NPC to be spawned, use the deactivate flag, DS3_DI1, to deactivate the spawn point after one NPC has been spawned.

There are three variables you may want to know about for future use that track the number of spawned children. nChildrenSpawned can be obtained with the statement:

```
int nChildrenSpawned = GetLocalInt(oSpawn, "ChildrenSpawned");
```

It tracks the number of children that spawn waypoint oSpawn has ever spawned, cumulatively. Note that the R subflag of PCR will reset this.

nSpawnNumber is the number of children oSpawn will maintain and is usually equal to the parameter attached to the SN flag, or one if no SN flag was included. If the Mm subflag was used with the SNn flag, then it will be a number between m and n. nSpawnNumber can be obtained with the statement:

```
int nSpawnNumber = GetLocalInt(oSpawn, "f_SpawnNumber");
```

nSpawnCount is the current number of spawned children of oSpawn. It can be obtained with the statement:

```
int nSpawnCount = GetLocalInt(oSpawn, "SpawnCount");
```

Want to turn off all spawn waypoints in an area that will not be used again? Use this script on the OnExit event of the area:

```
#include "spawn_functions"
void main() { DeactivateAllSpawns(); }
```

To summarize:

- you have a few functions you can use in your script files to access the spawn points and affect the creatures actions.

Chapter 28 - Spawn Banners

In this chapter we will:

- learn what the spawn banner system is,
- learn how to install the spawn banner system, and
- learn how to use it.

The spawn banner system is a tool for activating and deactivating the spawn points from within the game. These powers are only available in DM mode. Control of the spawn points is through a magic rod, the Rod of Spawn Banners. When used, the DM interacts with the spawn point via a conversation to change the settings of the spawn point.

Installing Spawn Banners

When you import the NESS spawn system into your module, you also get the spawn banner system. To install it, add the `spawnb_sample_ai` script to the module `OnActivateItem` event in the module properties dialog box as shown in Figure 65. Of course, it will only work in areas where you have installed the spawn system also.

Using Spawn Banners

Once in the game, as a DM, use the DM *Creator* to create the Rod of Spawn Banners. It is found in the custom *Creator > Item > Custom > Miscellaneous > Other* category as shown in Figure 66. Place the rod in your inventory. Right click on the rod and select *Activate*, then use the rod on the ground. This will spawn banners at the location of each spawn point in the area as shown in Figure 68. Clicking on the banner will display the spawn point properties. To make changes to the spawn point, right click on the rod, select *Activate*, and click on the banner. This will start a conversation with the spawn

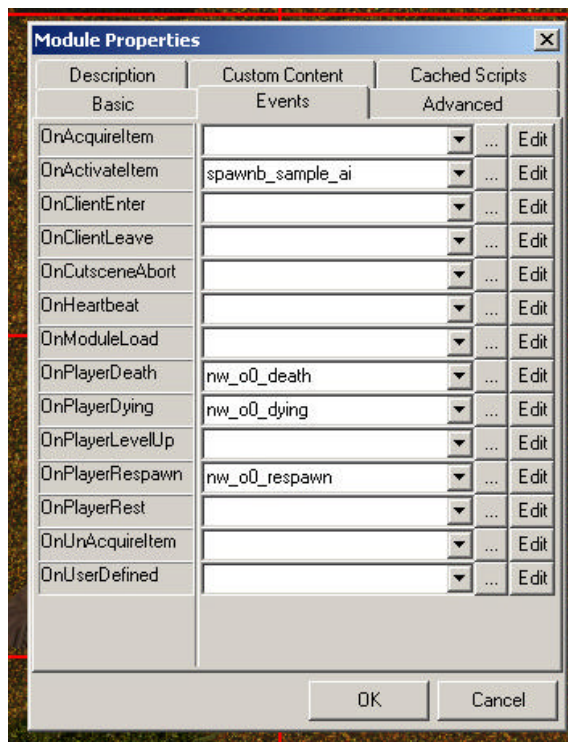


Figure 65 – Module properties dialog box.



Figure 66 – Create Rod of Spawn Banner.

point that will allow you to change its properties, as shown in Figure 67. To hide all the spawn banners in the area, activate the rod on the ground again. Note that the banners are displayed only at waypoints that are spawn waypoints, i.e. where the name field of the tag starts with an SP. Entrance, exits, patrol routes, and spawn location waypoints will not display banners.

The conversation with the spawn banner will allow you to do four things: activate or deactivate the spawn point, turn on or off spawn tracking, turn on or off spawn dumping, and you may debug NESS. The conversation also shows the flags in the name field of the waypoint and the creatures or placeables listed in the tag field of the waypoint.



Figure 68 – Spawn banners.

Activating or deactivating applies to the specific spawn point where the banner is located. You must activate and deactivate the spawn points individually.

Turning spawn tracking on and off is global to the area. It may be turned on at any banner and turned off at any banner in the area. When spawn tracking is on, a message is sent to all DMs whenever the total spawn count in an area changes. It reports the area that changed, the new spawn count in that area, and the new spawn count for the entire module.

Turning spawn dumping on and off is global to the area also. It may be turned on at any banner and turned off at any banner in the area. Spawn dumping will provide a report similar to spawn tracking for every area that has a non-zero spawn count each heartbeat.

Debugging NESS is global to the area also. It may be done at any banner in the area.



Figure 67 – Spawn banner conversation.

To summarize:

- use the spawn banner system in DM mode to change the functions of the spawn point and track the spawn count.

Appendix A – GNU Free Documentation License

GNU Free Documentation License
Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation,
Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307
USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an

otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified

Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will

automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix B – Appropriate Parameter Ranges for Flags

```

Flag (Range)[operator]Optional Flag(Range) Note

SP (01-99) SP00 = SP01
ID (00-99)
FT (00-99)
SN (01-99) |> M (00-99) SN00 = Not set
SA | (02-99) SA00 = SA
SA01 = Default (One Child per Cycle)
SR (01-99) |> M (00-99) SR00 = Not set
SD (01-99) |> M (00-99) SD00 = Not set
CL (01-99) |> M (00-99) CL00 = Not set
DY[01-28]|T[01-28]
HR[00-23]|T[00-23]
DO | D
NO | D
RW | R (01-99) RWR00 = Not set
PC PCR = Don't work
PC | (00-99) | R PC00 = PC
RG (001-999) | C (01-99) RG000 = Not set
RGxxxC00 = Not set
SF
SU (01-99) SU00 = Not set
FX (001-999) | D(001-999) FX000 = Dummy Effect - Never Use
FXxxxD000 = Dummy Effect - Never Use
PR (01-99) | T (00-03) PR00 = ??? Route Tag :
PR (01-99) PR00 = ???
SN (01-99) SN00 = ???
PS (001-999) PS000 = Not set
RN
SF
DO
NO
SC (000-999)
EE (00-99) | R (00-99) R flag is the other EE tag
EE (00-99)
EX (00-99) | R (00-99) R flag is the other EX tag
EX (00-99)
PL (00-02) | T (00-99) PLxxT00 = Trap Enabled
SG
CD (001-999) | T (00-03) | R (00-07)
LT (00-99)
DS (00-04) | S (000-999)
DI (01-99) Needed for DS (03-04)
DI00 = Not set
CM | D (000-999)
Camp Children Flags:
RW
SF
LT (00-99)
CD (000-999) | T (00-03)
PL (00-02) | T (00-99) PLxxT00 = Trap Enabled
SS (001-999) | D (001-999) SS000 = Dummy Script - Never Use
DT (001*-999) do not use DT000
CP (00-99) | R (01-99) CPxxR00 = Useless
CC (00-99)
TR (01-99) | D (01-99) TR00 = Useless
TRxxD00 = Useless
AE (001-999) | D (000-999) AE000 = Dummy AreaEffect - Never Use
AExxxD000 = Permanent Area Effect
OE (001-999) | D (000-999) OE000 = Dummy ObjectEffect - Never Use
OExxxD000 = Permanent Object Effect
RS (01-99) RS00 = ??? ( 0 % or 100 % ? )
FC (00-04)
AL (00-05) | S (01-99) AlxxS00 = Useless
HB (000-999)
SL (00-99) | R (00-99) R flag is the other SL tag

```

Location Tag:

SL00

RH | D (01-99) RHD00 = Useless

HL

IT

ST

PT

CF

by kuff

Appendix C – spawn_readme File

```
//
//
//  NESS
//  Version 8.0
//  Spawn_Readme
//
//
//  Neshke Narovken (Original Author)
//
//  Cereborn (Maintainer)
//
//*****
//
// History:
//
// +++ Start Version 7.0
//
//   --/--/--   Neshke           Created
//
//
// +++ Start Version 7.0.1
//
//   12/03/02   Cereborn         Added DanielleB's merchant-based loot tables
//   12/12/02   Cereborn         Added area-wide count of currently spawned
//                                   creatures
//   12/31/02   Cereborn         Added LT subflags A, B, and C for specifying
//                                   percentage chances of receiving 1, 2, or 3
//                                   loot items.
//   01/01/03   Cereborn         Added SX flag for dim returns suppression.
//
// +++ Start Version 7.0.2
//
//   01/07/03   Cereborn         Added NESS_ActivateSpawn(),
//                                   NESS_DeactivateSpawn, NESS_GetSpawnByID(),
//                                   NESS_ActivateSpawnByID(), NESS_ForceProcess()
//                                   NESS_DeactivateSpawnByID() and
//                                   NESS_TrackModuleSpawns().
//                                   Modified so that an activated spawn is processed
//                                   immediately instead of waiting for the next
//                                   process tick when using SPxx
//                                   Moved ReportSpawns and TrackModuleSpawns
//                                   to spawn_functions from spawn_main; modified
//                                   spawn_main to call TrackModuleSpawns() if
//                                   enabled, removing the need for placing
//                                   in each area heartbeat script
//
//   01/08/03   Cereborn         Fixed bug preventing initial flag processing
//                                   when using SPxx.
//                                   Added new deactivate spawn (DS) condition
//                                   (6) which deactivates a spawn whenever spawn
//                                   count reaches spawn number. This is similar
//                                   to DS1, except that DS1 is based on number
//                                   of children *ever* spawned, so once
//                                   deactivated can never be reactivated, while
//                                   DS6 is based on current child count, allowing
//                                   the spawn to be re-activated if 1 or more of
//                                   it's children have been killed or despawned
//   01/10/03   Cereborn         Prevent despawning creatures when possessed
//                                   by a DM. (the code was attempting to do so
//                                   already but due to a Bioware bug it wasn't
//                                   working)
//   01/11/03   Cereborn         Added I subflag to SU to check each child's
//                                   location individually instead of the spawn's
//                                   location
//   01/17/03   Cereborn         Added force process of spawns when PCs
//                                   re-arrive id deactivated by PC flag
//
```

```

// 01/18/03 Cereborn Added new type (3) to PL and subflag P
// Added P subflag to SD.
// 01/20/03 Cereborn Force process on deactivate
// Added NESS_IsModuleSpawnTracking()
//
// +++ Start Version 7.0.3
//
// 01/22/03 Cereborn Bug fix: Initialize module spawn count to
// 0 to fix bug with loading from saved games
// 01/24/03 Cereborn Added NESS_DumpModuleSpawns() and
// NESS_IsModuleSpawnDumping(). When spawn
// dumping is enabled, each area with active
// spawns reports its spawn count each HB
//
// +++ Start Version ALFA 1.0
//
// 02/05/03 - 03/21/03
//
// Added NESS_ProcessDeadCreature(). This can be used to notify NESS that
// a spawned creature has died (typically from an onDeath event script).
// This is particularly useful when using larger SPxx values and the CD
// flag, since without notification it can take a long time for NESS to
// discover that the spawned creature has died and set up the lootable
// corpse.
//
// Added O(offset) subflag to the SP flag. NESS now processes all spawns on
// the first heartbeat, and then spawns are be processed on heartbeat
// 1 + offset + (processRate * N). For example, SP3 gets processed on
// heartbeats 1, 4, 7, 10, etc.... while SP301 gets processed on heartbeats
// 1, 5, 8, 11..., and so on. This should be used to spread out heartbeats
// within an area.
//
// NESS no longer does any timing based on counted heartbeats. All timings
// are now based on the clock. A new file, spawn_timefuncs, contains
// functions for converting the game clock to real seconds for use in NESS
// timings.
//
// When spawns despawn due to using the PC flag (and if the R subflag is
// not specified) the despawning creature resrefs and their locations are
// recorded, and the spawn is restored to it previous state when PCs
// re-enter the area. This prevents exploits where players intentionally
// 'pop' a zone, then leave for the PC flag's duration, to effectively
// clear an area of all obstacles until the spawn delay timer (if any)
// expires.
//
// Fixed the bug causing corpses that had been fully looted to not decay.
//
// Fixed the bug that caused corpses to *never* decay if the decay timer
// went off when the inventory was open (now a new decay timer is set).
//
// Flag parsing code cleaned up; fixed a bug that caused the default value
// for a flag to always be 1 (ignoring what was set up in spawn_cfg_global)
//
// Items marked no-drop are no longer copied onto the lootable corpses
//
// All no-drop items are explicitly destroyed (whether the CD flag is used
// or not) to working around a current Bioware memory leak.
//
// Added NESS_ReturnHome(). Call to force a NESS spawn to it's home point.
//
// Added support for RH flag on spawn camps.
//
// +++ Start Version ALFA 1.0 Patch 1 (v1.0.1)
//
// 03/29/03
//
// A bug where spawn delays were being applied to first time spawns that
// didn't happen right away (such as triggered spawns, day/night only,
// etc.) was fixed.

```



```
//  
//      If a spawn has the RH flag and is respawned after being despawned due  
//      to the PC flag, creatures will respawn at their home point instead of  
//      where they were when they despawned.  
//  
//      An errant debug statement was removed.  
//  
// +++ Start Version ALFA 1.0 Patch 2 (v1.0.2)  
//  
//      03/30/03  
//  
//      Fixed a bug where the number of creatures in a spawn was getting  
//      confused due to spawns that use the M subflag of the SN flag  
//      recalculating their spawn numbers when spawns deactivated due to the  
//      PC flag. This bug fix should eliminate the overpopulating spawns  
//      problem.  
//  
// +++ Start Version ALFA 1.0 Patch 3 (v1.0.3)  
//  
//      03/31/03  
//  
//      Fixed a bug where spawn delays could be skipped.  
//  
// +++ Start Version ALFA 1.0 Patch 4 (v1.0.4)  
//  
//      04/04/03  
//  
//      Fixed a bug where spawn delays were ignored on placeables.  
//      Fixed P subflag of SD being ignored.  
//      Reset spawn delay after RS-failed spawn attempt  
//      Set up a new spawn delay when creature killed  
//  
// +++ Start Version ALFA 1.0 Patch 5 (v1.0.5)  
//  
//      04/07/03  
//  
//      Fixed bug in spawn_timefuncs that resulting in incorrect conversions  
//      in years other than the Epoch year.  
//      Changed the Epoch year to 1340 to conform to the lowest date in  
//      the NWN engine.  
//  
// +++ Start Version ALFA 1.0 Patch 6 (v1.0.6)  
//  
//      04/08/03  
//  
//      Fixed bug with corpse decay and death script flags not being available  
//      for camp spawns when onDeath notification occurs.  
//  
// +++ Start Version ALFA 1.0 Patch 7 (v1.0.7)  
//  
//      04/27/03  
//  
//      Added debugging for spawn delays and spawn counts that can be  
//      independently enabled/disabled for each area from the spawn banner  
//      rod.  
//  
// +++ Start Version ALFA 1.1 (v1.1)  
//  
//      04/28/03  
//  
//      Fixed bug in Loot Merchant code. The original code used  
//      GetNearestObjectByTag() to look up loot merchants, which according to  
//      the documentation should never have worked for merchants not in the same  
//      area as the creature spawned. In reality, it stopped working (at least  
//      in some cases) after Bioware released version 1.29 of the game.  
//  
//      05/03/03  
//  
//      Changed distribution method used for SR flag to evenly distribute  
//      spawns in the spawn circle instead. The old method made the spawns  
//      denser near the center and rarer near the circle's edge
```

```
//  
//      Made SF work (again?) for placeables  
//  
//      When the SF flag is not specified for a multi-child spawn, a random  
//      SF is now calculated independently for each child  
//  
//      05/08/03  
//  
//      Added NL (No Loot) flag. This suppresses looting of player corpses.  
//      Only applies when using ACR 1.14 and higher  
//  
//      05/25/03  
//  
//      Fixed overspawning bug caused by changing child counts on spawns that  
//      had been 'saved' with potentially different counts  
//  
//      05/27/03  
//  
//      Added C (Closest) subflag to the PR (Patrol Route) flag. If C is  
//      specified, the spawned creature will start at the closest waypoint  
//      rather than the first (does not apply to T2 flagged routes(random  
//      traverse)).  
//  
//      05/30/03  
//  
//      Added SB (Subdual) flag. Causes creatures to be spawned in in subdual  
//      mode. Only applies when using ACR 1.14 and higher.  
//  
//  
// +++ Start Version ALFA 1.2 (v1.2)  
//  
//      08/24/03  
//  
//      Removed LocationToString() function from spawn_functions, as this is  
//      now a Bioware function  
//  
//      08/25/03  
//  
//      Modified the way the SX flag works. First, there is now a global flag,  
//      nGlobalSuppressDR, in spawn_cfg_global that can be set to determine  
//      whether or not creatures spawn in with DR on or off when no SX flag is  
//      specified. The current default is for DR to *not* be suppressed, i.e.,  
//      it will be operational. Also, the SX flag can now take a value of 0 or 1.  
//      If 1, DR is suppressed, if 0 DR is enabled (useful if you've set  
//      nGlobalSuppressDR to 1). The default if no value is specified (just SX)  
//      is 1 (to suppress). You can also change this default in spawn_cfg_global.  
//      If nGlobalSuppressDR is 1 and nSuppressDR is 0, you will get the opposite  
//      of the v1.1 functionality; no DR on creatures, except when the SX flag is  
//      present.  
//  
// +++ Start Version ALFA 1.2.1 (v1.2.1)  
//  
//      09/01/03  
//  
//      Fixed bug where spawn number was always being set to the number of saved  
//      camp spawns on PC-flag restore instead of the sum of saved camp and  
//      regular spawn counts.  
//  
// +++ Start Version ALFA 1.2.2 (v1.2.2)  
//  
//      09/02/03  
//  
//      Fixed bug introduced by last bug fix that caused overspawning!  
//  
// +++ Start Version ALFA 1.2.3 (v1.2.3)  
//  
//      09/21/03  
//  
//      Added EL - E(ncounter) L(evel) flag  
//  
// +++ Start Version ALFA 1.2.4 (v1.2.4)
```

```
//
// 10/03/03
//
// Added I subflag to SL flag.
//
// Modified ST behavior so that spawned creatures walk to their seats
// instead of running
//
// 10/18/03
//
// Check night / day only, day, hour, lifespan and SU before restoring
// spawns
//
// Fixed recalculate random spawn number bug where the spawn number could
// change before a despawn due to the PC flag was restored, causing the
// number of creatures thought to exist to differ from what actually got
// restored
//
// 10/19/03
//
// Fixed SD bug; under certain cases (such as a despawn due to CL flag)
// SD was being ignored.
//
// Fixed bug where SU|I only worked if RS or SL were in use
//
// +++ Start Version ALFA 1.2.5 (v1.2.5)
//
// 10/25/03
//
// Do a ClearAllActions() before despawning creatures. This helps prevent
// 'broken' Bioware chairs caused by despawning creatures using the ST
// flag.
//
// 10/27/03
//
// Modified the spawn_cfg_camp example to use standard BW creatures /
// placeables for the benefit of non-ALFA users.
//
// 11/02/03
//
// Added a scaled encounter example. This uses the same basic methodology
// as Sareena's random wilderness spawns - an SNxx flag is given large
// enough for the largest possible spawn and then the actual number (and
// types) of creatures is determined when the spawn actually takes place.
//
// +++ Start Version 8.0
//
// Given that NESS is now being supported outside the ALFA umbrella, I've
// decided to begin version numbering from 8.0 from here out. Version 8.0
// is the immediate successor to ALFA version 1.2.5.
//
// 01/19/04
//
// Fixed problems with naked NPC corpses that had droppable armor /
// clothing. playable race characters (humans, dwarves, elves, etc.)
// now keep a copy of whatever is in the chest slot on the original
// corpse.
//
// 01/21/04
//
// Added Rn subflag to CD to specify what type of remains are left after
// corpse decay. These correspond 1 to 1 to the treasure type field for
// placeables, except for R7, which causes no loot bag to be left (loot
// destroyed when corpse decays). See the specific flag documentation
// below for the values/types of each R subflag.
//
// Added D subflag to CD to cause corpse to drop wielded weapons on the
// ground. Note that droppable flag on weapon still takes precedence -
```

```

//      non-droppable wielded weapons will not be dropped.
//
//      Delete armor/clothing from corpse if looted.
//
//      Added scripts for lootable corpse onOpen, onClosed, onUsed, and
//      onDisturbed events (renamed with a spawn_ prefix).
//
//      Added a lootable corpse placeable for each remains type; each has the
//      correct event scripts attached.
//
//      01/22/04
//
//      Initialize global defaults and flags on first area heartbeat, not first
//      heartbeat with PCs present.
//
//      Use DelayCommand(0.0, ...) to give each spawn flag initialization
//      function its own command queue (allowing many more spawn points before
//      TMIs at initialization occur).
//
//      01/23/04
//
//      Fixed bug with EE flag. Spawned creature was not walking to spawn point
//      after entrance.
//
//      Fixed bug with spawn in effect when using EE. Spawn in effect now happens
//      at the entrance point.
//
//      Added support for ALFA-specific flags as Custom Flags (following the CF
//      flag. Parsing and processing of custom flags can now be done by
//      modifying spawn_cfg_cusflg. This file contains 2 functions:
//      ParseCustomFlags() and SetupCustomFlags(). ParseCustomFlags is called
//      with whatever flags follow the CF_ flag (when flags for the spawn are
//      being initialized. Typically, flags are parsed and their values are
//      written to the spawn object. SetupCustomFlags() is called when a
//      creature is actually spawned (typically flags are copied from the spawn
//      object to the creature (spawned) object. The processing of ALFA-specific
//      flags are included in this file as an example.
//
//
//      INCLUDES:
//
//      Do NOT Modify Main Script:
//
//      Spawn:          'spawn_main'
//
//      Do NOT Modify Helper Scripts:
//
//      Information:    'spawn__readme'
//      Functions:      'spawn_functions'
//      Corpse Death:   'spawn_corpse_dth'
//      Corpse Decay:   'spawn_corpse_dcy'
//      Global Defaults: 'spawn_defaults'
//      Flag Parsing:   'spawn_flags'
//
//      Lootable corpse event scripts:
//
//      Corpse On Open: 'spawn_onopencrp'
//      Corpse On Closed: 'spawn_oncloscrp'
//      Corpse disturbed: 'spawn_dist_corps'
//      Corpse used:     'spawn_used_corps'
//
//      CONFIGURATION:
//
//      Modify Configuration Includes if Needed
//      Only where Specified in Each File:
//
//      Spawn Flags:    'spawn_cfg_flag'
//      Spawn Groups:   'spawn_cfg_group'
//      Loot Tables:   'spawn_cfg_loot'

```

```

//      Spawn Camps:      'spawn_cfg_camp'
//      Spawn Effects:    'spawn_cfg_fxsp'
//      Area Effects:     'spawn_cfg_fxae'
//      Object Effects:   'spawn_cfg_fxobj'
//      User Defaults:    'spawn_cfg_global'
//      Custom Flags:     'spawn_cfg_cusflg'
//
// Modify Check Includes if Needed
// Only where Specified in each File:
//
//      Check PCs:        'spawn_chk_pcs'
//      Custom Check:     'spawn_chk_custom'
//
// Modify Scripting Includes if Needed
// Only where Specified in Each File:
//
//      Spawn/Despawn:    'spawn_sc_spawn'
//      Heartbeat:        'spawn_sc_hbeat'
//      Death:            'spawn_sc_death'
//      Deactivation:     'spawn_sc_deactiv'
//      Camp Trigger:     'spawn_sc_cmptrig'
//      Patrol Stops:     'spawn_sc_patrol'
//
//
// USAGE:
//
// Add Following to Area Heartbeat:
//
//      Sample Script:    'spawn_sample_hb'
//
//      #include "spawn_main"
//      Spawn();
//
// Add Spawn Waypoints
// All Switches Optional
//
// Waypoint Name:
//      SP_<Spawn Flags>
//
// Waypoint Tag:
//      Child Object Tag/ResRef
//      Group Designation
//      Camp Designation
//
// Note:
//      For Custom Creatures
//      You must use the ResRef
//
// Available Flags:
//      SPnOn_ISnDn_IDn_FTn_SNnMn_SAnMn_SRnMnP_SDnMn_CLnMn
//      _RGnMnCn_RWRn_DYnTn_HRnTn_DOD_NOD_PCnR_SF_SUnIn_FXnDn
//      _PRnTn_PLnTn_EEnRn_EXnRn_SG_CDnTnRnD_LTnAnBnCn_DSnSn_DIn_CMDn
//      _SSnDn_CCn_CPnRn_TRnDn_AEnDn_OEnDn_RS_n_FCn
//      _ALnSn_HBn_SLnRnI_RHDn_DTn_HLnE_IT_ST_PT_SM_CF
//      _SXn_NL_SB_ELn
//
//
// Format:
//      Flag|OptionalFlag
//
//
//      SPn|On
//      : Designates Spawn Waypoint
//      : SP is Required on all Spawn Waypoints
//      : Performs Spawn processing every SP1 to SP99 heartbeats
//      : Default is SP1 Heartbeat : ~6 Seconds
//      : SP00 Defaults to SP01
//      : Optional Flag: On
//      : Offset processing spawn from first process by n heartbeats.
//      : Used to stagger spawn processing within an area
//

```

```

// ISn|Dn
// : Initial State
// : Type 0 : Inactive
// : Type 1 : Default : Active
// : Optional Flag: D000
// : Delay Initial Spawn D000 Minutes
//
// IDn
// : SpawnID
// : Sets LocalInt "SpawnID" to SpawnID
// : On Spawn Waypoint Object
//
// FTn
// : Flag Table
// : Use Flags from Table FT00
// : Flags are Defined in 'spawn_cfg_flag'
//
// SNn|Mn
// : Spawn Number
// : Maintains a Spawn of SN00 to SN99 Children
// : Despawns Extra Children
// : Optional Flag: M00
// : Minimum Children
// : Will Randomly Spawn between M00 and SN00 Children
//
// SA|n|Mn
// : Spawn All at Once
// : Default is Spawn One Child per Cycle
// : Optional Flag: 00
// : Spawn 00 Children per Cycle
// : Optional Flag: M00
// : Spawn Minimum M00 per Cycle
//
// SRn|Mn|P
// : Spawn Radius
// : Randomly Spawns Children in a
// : Location SR00 to SR99 Meters from Waypoint
// : Optional Flag: M00
// : Minimum Radius, M00 to SR99 Meters
// : Optional Flag: P
// : Spawn Center is Near a Random PC in Area
//
// SDn|Mn|P
// : Spawn Delay
// : Delay SD00 to SD99 Minutes between Spawns
// : Optional Flag: M00
// : Minimum Delay, M00 to SD99 Minutes
// : Optional Flag: P
// : This flag sets up a spawn period, whereas the default spawn delay
// : specifies a spacing between spawns.
// : This is particularly useful with PL3 (keep the times the same on
// : each) it will cause the placeable to refresh if alive or respawn
// : if not on a regular schedule. Note that if the P subflag is used,
// : the M subflag is ignored.
//
// CLn|Mn
// : Child Lifespan
// : Child will Despawn after CL00 Minutes
// : Optional Flag: M00
// : Child will Despawn after M00 to CL00 Minutes
//
// DYn|Tn
// : Spawn Day
// : Spawn Only on Day DY00 to DY28
// : Children are Despawnd during Invalid Days
// : Optional Flag: T00
// : Spawn from Day DY00 to Day T00
//
// HRn|Tn

```

```

// : Spawn Hour
// : Spawn Only during Hour HR00 to HR24
// : Children are Despawned during Invalid Hours
// : Optional Flag: T00
// : Spawn from Hour HR00 to Hour T00
//
// DO|D
// : Day Only
// : Only Spawns at Day
// : Optional Flag: D
// : Despawn Children at Nightfall
//
// NO|D
// : Night Only
// : Only Spawns at Night
// : Optional Flag: D
// : Despawn Children at Daybreak
//
// RW|Rn
// : Random Walk
// : Children Wander Randomly
// : Optional Flag: R00
// : WARNING: This is Resource Intensive!!
// : Wander Range, R00 to R99 Meters
//
// PC|n|R
// : PC Check
// : Only Spawn Children if PCs are in Area
// : Children are Despawned if no PCs in Area
// : Optional Flag: PC00
// : Depawn if no PCs in Area for PC00 Minutes
// : Optional Flag: R
// : Reset Spawn Point if no PCs are Present
//
// RGn|Mn|Cn
// : Random Gold
// : Generates Random Amount of Gold on Children
// : From RG000 to RG999 Gold
// : Optional Flag: M00
// : Minimum Gold Amount
// : Optional Flag: C00
// : Gold Chance C00% to C99%
// : Default 100% chance
//
// SF
// : Spawn Facing
// : Set Facing of Children to Match Waypoint
// : Default Random Facing
//
// SUn|In
// : Spawn Unseen
// : Only Spawn if PCs are not within SU00 to SU99 Meters
// : Optional flag: I00
// : Use the location of each individual child to determine if that
// : child can spawn, instead of using to location of the spawn itself.
// : If using SR or SL|R, I00 to I99 additional random locations will be
// : attempted
//
// FXn|Dn
// : Spawn Effect
// : Spawn in with Effect FX001 to FX999
// : Effects are Defined in 'spawn_cfg_fxsp'
// : Optional Flag: D000
// : Despawn with Effect D001 to D999
//
// PRn|Tn|C
// : Patrol Route
// : Assign Waypoints Route PR00 to PR99
// : Optional Flag: T0
// : Route Type
// : 0 - Sequential

```

```

// : 1 - Circular
// : 2 - Random
// : 3 - Walk Once/Despawn
// : Optional Flag C
// : Start at closest waypoint. Does not apply to T2 (random)
// :
// : Route Name: Variable
// : Route Tag: PR00_SN00_PS000_RN_FC_DO_NO_SC000
// :
// : PR - Route Number 00 to 99
// : SN - Stop Number 00 to 99
// : PS - Pause 000 to 999 seconds at Stop
// : RN - Run to Stop
// : SF - Face the Waypoint Direction
// : DO - Only Stop here during the Day
// : NO - Only Stop here during the Night
// : SC - Run script 000 to 999
// : Scripts are Defined in 'spawn_sc_patrol'
//
// EE|n|Rn
// : Entrance/Exit
// : Enter and Exit at Waypoint EE00
// : Optional Flag: R
// : Choose Random Entrance Exit
// : from R00 to EE00
// :
// : Entrance/Exit Name: Variable
// : Entrance/Exit Tag: EE00
//
// EX|n|Rn
// : Exit
// : Exit at Waypoint EX00
// : Optional Flag: R
// : Choose Random Exit
// : from R00 to EX00
// :
// : Exit Name: Variable
// : Exit Tag: EX00
//
// PL|n|Tn|Pn
// : Placeable Object
// : Spawns a Placeable Object with Behavior
// : Behavior 0: Default Behavior
// : Behavior 1: Despawn if Empty
// : Behavior 2: Refill if Empty (after spawn delay minutes!)
// : Behavior 3: Refresh (despawn/respawn) every P000 minutes; default
// : (if no Pn specified) is 60 minutes
// : Optional Flag: T00
// : Trap Disabled Chance
// : 00% to 99% chance of Trap Disabled
// : Trap must already be part of Placeable's Template
// : Default 100% Chance Trap is Disabled
// : Optional Flag: P00
// : Refresh period (in minutes)
//
// SG
// : Spawn Group
// : Spawn Children from Group
// : Waypoint Tag is Defined Group
// : Groups are Defined in 'spawn_cfg_group'
//
// CD|n|Tn|Rn|D
// : Corpse Decay
// : Decay Corpse after CD000 to CD999 Seconds
// : Default No Corpse, Standard Loot Bag
// : Optional Flag: T
// : Corpse Inventory Type
// : Type 0: Inventory Items
// : Type 1: Inventory & Equipped Items
// : Type 2: Inventory Items, if PC Killed
// : Type 3: Inventory & Equipped Items, if PC Killed

```



```

// : Optional Flag: R
// : Remains Type
// : Type 0: Loot bag
// : Type 1: Body
// : Type 2: Bones
// : Type 3: Potion
// : Type 4: Pouch
// : Type 5: Scroll
// : Type 6: Treasure
// : Type 7: None (destroy loot on decay)
// : Optional Flag: D
// : Drop wielded weapons
//
// LTn|An|Bn|Cn
// : Loot Table
// : Spawn Loot on Children from Table LT000 to LT999
// : Loot Tables are Defined in 'spawn_cfg_loot'
// : LT500 to LT999 currently reserved for merchant-based
// : loot tables
// : Optional Flag: A000
// : When using merchant-based loot tables, the percentage chance that
// : only 1 item will spawn. Default is 50%. Values over 100% are truncated
// : to 100%
// : Optional Flag: B000
// : When using merchant-based loot tables, the percentage chance that
// : 2 items will spawn. Default is 15%. Values over 100% are truncated
// : to 100%
// : Optional Flag: C000
// : When using merchant-based loot tables, the percentage chance that
// : 3 items will spawn. Default is 05%. Values over 100% are truncated
// : to 100%
//
// DSn|Sn
// : Deactivate Spawn
// : Deactivate Spawn based on Condition
// : Type 0: Deactivate if all Children are Dead
// : Type 1: Deactivate if Spawn Number has been Spawned
// : Type 2: Deactive Spawn until all Children are Dead
// : Type 3: Deactivate Spawn after DI00 Children
// : Type 4: Deactivate Spawn after DI00 Minutes
// : Type 5: Deactivate Spawn after DI00 Cycles
// : Type 6: Deactivate when Spawn Count == Spawn Number
// : Optional Flag: S000
// : Run Script 000 to 999 when Spawn Deactivated
// : Scripts are Defined in 'spawn_sc_deactiv'
// :
// : Can Reactivate by Manually Calling:
// : SetLocalInt(oSpawn, "SpawnDeactivated", FALSE);
//
// DIn
// : Deactivation Information
//
// CM|Dn
// : Spawn Camp
// : Spawns Camp of Creatures and Placeables
// : Waypoint Tag is Defined Camp
// : Despawns Camp when all Children Dead
// : Camps are Defined in 'spawn_cfg_camp'
// : Optional Flag: D000
// : Placeables Decay 000 Seconds after Camp Despawn
// :
// : Camp Children Flags:
// :
// : RW : Random Walk
// : SF : Spawn Facing Camp
// : SG : Spawn Group
// : LT00 : Loot Table
// : CD000|T0 : Corpse Decay
// : PL0|T00 : Placeable Type
// : Placeable Trap Disabled
//

```

```

//  SSn|Dn
//  : Spawn Script
//  : Run Script 001 to 999 on Spawn
//  : Scripts are Defined in 'spawn_sc_spawn'
//  : Optional Flag: D000
//  :   Run Script 001 to 999 on Despawn
//
//  DTn
//  : Death Script
//  : Run Script 001 to 999 on Death
//  : Scripts are Defined in 'spawn_sc_death'
//
//  CPn|Rn
//  : Spawn Check PCs
//  : Check Custom Code to see if Spawn Proceeds
//  : Custom Code is Defined in 'spawn_chk_pcs'
//  : Optional Flag: R00
//  :   Check all PCs in Radius R00
//  :   Default Check all PCs in Area
//
//  CCn
//  : Spawn Check Custom
//  : Check Custom Code to see if Spawn Proceeds
//  : Custom Code is Defined in 'spawn_chk_custom'
//
//  TRn|Dn
//  : Spawn Trigger
//  : Only Spawns if PC is within 00 Meters
//  : Optional Flag: D00
//  :   Despawns if PC is not within D00 Meters
//
//  AEn|Dn
//  : Spawn Area Effect
//  : Area Effects are Defined in 'spawn_cfg_fxae'
//  : Waypoint Tag can be "AE" to Spawn only Area Effect
//  : Optional Flag: D000
//  :   Area Effect Duration of 000 to 999 Seconds
//  :   Duration of 000 means Permanent Area Effect
//  :   Default Duration is 005 Seconds
//
//  OEn|Dn
//  : Object Effect
//  : Object Effects are Defined in 'spawn_cfg_fxobj'
//  : Optional Flag: D000
//  :   Object Effect Duration of 000 to 999 Seconds
//  :   Duration of 000 means Permanent Object Effect
//  :   Default Duration is Permanent
//
//  RSn
//  : Random Spawn
//  : Percentage Chance Spawn will Occur
//  : Default 100% Chance
//
//  FCn
//  : Spawn Faction
//  : Change Faction of Children:
//  :   Faction 0: COMMONER
//  :   Faction 1: DEFENDER
//  :   Faction 2: MERCHANT
//  :   Faction 3: HOSTILE
//  :   Faction 4: CUSTOM
//  :   Change Faction to Same as Nearest
//  :   Object with Tag 'SpawnFaction'
//
//  ALn|Sn
//  : Spawn Alignment
//  : Shift Alignment of Children
//  :   Alignment 0: Neutral
//  :   Alignment 1: Law
//  :   Alignment 2: Chaos
//  :   Alignment 3: Good

```

```

// : Alignment 4: Evil
// : Alignment 5: All
// : Optional Flag: S00
// : Shift Alignment by S00
// : Default Shift by 10
//
// Hbn
// : Heartbeat Script
// : Children will Run Script HB000 each Cycle
// : Scripts are Defined in 'spawn_sc_hbeat'
//
// SLn|Rn|I
// : Spawn Location
// : Spawn Children at Waypoint SL00
// : Optional Flag: R
// : Choose Random Location
// : from R00 to SL00
// : Optional Flag: I
// : When spawning multiple children, each child takes the next
// : waypoint, In order - (first child spawns at SL00, next at SL01, etc.)
// :
// : Location Name: Variable
// : Location Tag: SL00
//
// RH|Dn
// : Return Home
// : Child will always Return to Home
// : Optional Flag: D00
// : Child will Return to Home
// : Only if further than D00 Meters
//
// HL|n|E
// : Heal Children if Not in Combat
// : Optional Value: 00
// : Heal 00% per Cycle
//
// IT
// : Spawn Item
// : Spawnpoint Tag is Item Template
//
// ST
// : Spawn Sit
// : Children will Sit in Nearest Unoccupied Sittable
// : Sittable's Tag must be 'Seat'
//
// PT
// : Spawn Plot
// : Sets Children as Plot
//
// SM
// : Spawn Merchant
// : Spawnpoint Tag is Merchant Template
//
// CF
// : Custom Flag
// : Everything in Spawn Name after CF
// : Is Stored in LocalString "CustomFlag"
// : On Each Spawned Child
//
// ALFA Only Flags
//
// SXn
// : Suppress XP
// : Suppress diminishing returns XP
// : SX1 turns suppression on (the default, you may just use SX)
// : SX0 turns suppression off for the spawn if it has been put on
// : globally (by setting nGlobalSuppressDR to TRUE in spawn_cfg_global)
//
// NL
// : No Loot
// : Suppress player corpse looting

```

```

//
// SB
// : SuBdual
// : Spawn creatures in in subdual mode
//
// ELn
// : Encounter Level
// : Set the encounter level for a spawn
// : This is used by the ALFA core rules in determining whether or not
// : an encounter should result in XP to a given level party. If not
// : specified, the CR of the creature killed is used as the EL.
//
//
//
//
// Functions for external control. Please use these when possible instead
// of modifying variables directly in NESS.
//
// object NESS_GetSpawnByID(int nSpawnID, object oArea)
// : Returns the spawn waypoint in area oArea with ID nSpawnID
//
// void NESS_ActivateSpawnByID(int nSpawnID, object oArea)
// : Activates the spawn in area oArea with ID nSpawnID
//
// void NESS_DeactivateSpawnByID(int nSpawnID, object oArea)
// : Deactivates the spawn in area oArea with ID nSpawnID
//
// void NESS_ActivateSpawn(object oSpawn)
// : Activates spawn oSpawn
//
// void NESS_DeactivateSpawn(object oSpawn)
// : Deactivates spawn oSpawn
//
// void NESS_ForceProcess(object oSpawn)
// : Force spawn oSpawn to be processed next heartbeat (regardless of
// : SPxx value)
//
// void NESS_TrackModuleSpawns(int nFlag=TRUE)
// : Enable (nFlag == TRUE) or Disable (nFlag == FALSE) Spawn Tracking
//
// int NESS_IsModuleSpawnTracking()
// : Returns TRUE is spawn tracking is enabled, FALSE otherwise
//
//
// Some Tips and Pointers
// - with Thanks to Eliha for Creating this Section
//
// 1. The .erf script files we import for NWN usually contain .nss and
// .utp files.
//
// 1.1. The .nss files are the Nwn Source Scripts that are in plain text that
// we manipulate in the script editor.
//
// 1.2. The .utp file(s) are objects such as an invisible corpse object that
// automatically gets put in the right column of the toolset where you select
// the creatures, doors, placeables, etc. under the custom button.
//
// 2. After importing the .erf and ignoring that it couldn't find the .ncs files
// you'll need to save your module then reopen it for the new scripts to appear
// in the left column (due to the lack of a refresh button).
//
// 3. When working in the various configuration scripts you only need to save
// them, not compile them. They are just add-ons to the main spawn_sample_hb.nss
// script and attempts to compile them will result in errors. The spawn_sc_xxxx
// files will compile but even they should just be saved like all the other
// spawn_cfg_xxxx scripts.
//
// 4. When finished editing and saving the various configuration scripts you
// will then need to open the spawn_sample_hb script and compile it to integrate

```

```
// your new changes into the system. If you customized your scripts, saved them,  
// saved the module, then loaded the game and find nothings working...chances are  
// you forgot to compile the spawn_sample_hb script after making your changes.  
//  
// 5. You can usually tell which script needs to be compiled by noticing which  
// one needs to be put in an area heartbeat or on an object somewhere, etc.. In  
// this case, we are adding the spawn_sample_hb to the area heartbeat so it is the  
// main file and requires compiling.  
//
```

```
//  
// Package Header  
//  
/*  
NESS Version 8.0
```

Cereborn
Legends Of Greyhawk

See 'spawn__readme' for Instructions
Ignore all 'Missing Resource' Errors
See the NWN Forums for More Info

Spawn Package:

```
NESS Scripts:  
- spawn__readme  
- spawn_cfg_camp  
- spawn_cfg_flag  
- spawn_cfg_fxae  
- spawn_cfg_fxobj  
- spawn_cfg_fxsp  
- spawn_cfg_global  
- spawn_cfg_group  
- spawn_cfg_loot  
- spawn_cfg_cusflg  
- spawn_chk_pcs  
- spawn_chk_custom  
- spawn_corpse_dcy  
- spawn_corpse_dth  
- spawn_defaults  
- spawn_dist_corps  
- spawn_flags  
- spawn_functions  
- spawn_main  
- spawn_onopencrp  
- spawn_onclocsrp  
- spawn_sample_hb  
- spawn_sc_cmptrig  
- spawn_sc_deactiv  
- spawn_sc_death  
- spawn_sc_patrol  
- spawn_sc_spawn  
- spawn_timefuncs  
- spawn_used_corps
```

```
NESS Resources:  
- invis_corpse_obj  
- invis_corpse_bdy  
- invis_corpse_bon  
- invis_corpse_pot  
- invis_corpse_pch  
- invis_corpse_ser  
- invis_corpse_tre
```

```
Spawn Banner Scripts:  
- spawnb_cc_activ  
- spawnb_cc_dactiv  
- spawnb_cc_dump  
- spawnb_cc_nodump
```

- spawnb_cc_notrck
- spawnb_cc_nsclog
- spawnb_cc_nsdlog
- spawnb_cc_sclog
- spawnb_cc_sdlog
- spawnb_cc_trck
- spawnb_main
- spawnb_sample_ai
- spawnb_sc_activ
- spawnb_sc_dactiv
- spawnb_sc_disp
- spawnb_sc_dump
- spawnb_sc_nodump
- spawnb_sc_notrck
- spawnb_sc_nsdlog
- spawnb_sc_sclog
- spawnb_sc_sdlog
- spawnb_sc_snclog
- spawnb_sc_trck

Spawn Banner Resources:

- spawn_ban_rod.uti
- spawn_banner.dlg
- spawn_ban_a.utp
- spawn_ban_d.utp

* /

Appendix D – Contents of NESS8_0Full.erf

NESS spawn system script files:

```
spawn__readme
spawn_cfg_camp
spawn_cfg_cusflg
spawn_cfg_flag
spawn_cfg_fxae
spawn_cfg_fxobj
spawn_cfg_fxsp
spawn_cfg_global
spawn_cfg_group
spawn_cfg_loot
spawn_chk_custom
spawn_chk_pcs
spawn_corpse_dcy
spawn_corpse_dth
spawn_defaults
spawn_dist_corps
spawn_flags
spawn_functions
spawn_main
spawn_oncloscrp
spawn_onopencrp
spawn_orig_hb
spawn_sample_hb
spawn_sc_cmptrig
spawn_sc_deactiv
spawn_sc_death
spawn_sc_hbeat
spawn_sc_patrol
spawn_sc_spawn
spawn_timefuncs
spawn_used_corps
```

NESS objects:

Placeables > Containers & Switches: corpse (seven of them)

Banner system files:

```
spawnb_cc_activ
spawnb_cc_dactiv
spawnb_cc_dump
spawnb_cc_nodump
spawnb_cc_notrck
spawnb_cc_nsclog
spawnb_cc_nsdlog
spawnb_cc_sclog
spawnb_cc_sdlog
spawnb_cc_trck
spawnb_main
spawnb_sample_ai
spawnb_sc_activ
spawnb_sc_dactiv
spawnb_sc_disp
spawnb_sc_dump
spawnb_sc_nodump
spawnb_sc_notrck
spawnb_sc_nsdlog
spawnb_sc_sclog
spawnb_sc_sdlog
spawnb_sc_snclog
spawnb_sc_trck
```

Banner system objects:

Items > Miscellaneous Other: Rod of Spawn Banners
Placeables > Penants & Signs: Active Spawn
Placeables > Penants & Signs: Deactivated Spawn

Banner system conversations:

spawn_banner

Appendix E – Visual Effect Constants

```

int VFX_NONE                = -1;
int VFX_DUR_BLUR            = 0;
int VFX_DUR_DARKNESS       = 1;
int VFX_DUR_ENTANGLE       = 2;
int VFX_DUR_FREEDOM_OF_MOVEMENT = 3;
int VFX_DUR_GLOBE_INVULNERABILITY = 4;
int VFX_DUR_BLACKOUT       = 5;
int VFX_DUR_INVISIBILITY   = 6;
int VFX_DUR_MIND_AFFECTING_NEGATIVE = 7;
int VFX_DUR_MIND_AFFECTING_POSITIVE = 8;
int VFX_DUR_GHOSTLY_VISAGE = 9;
int VFX_DUR_ETHERREAL_VISAGE = 10;
int VFX_DUR_PROT_BARKSKIN  = 11;
int VFX_DUR_PROT_GREATER_STONESKIN = 12;
int VFX_DUR_PROT_PREMONITION = 13;
int VFX_DUR_PROT_SHADOW_ARMOR = 14;
int VFX_DUR_PROT_STONESKIN = 15;
int VFX_DUR_SANCTUARY     = 16;
int VFX_DUR_WEB            = 17;
int VFX_FNF_BLINDDEAF     = 18;
int VFX_FNF_DISPEL        = 19;
int VFX_FNF_DISPEL_DISJUNCTION = 20;
int VFX_FNF_DISPEL_GREATER = 21;
int VFX_FNF_FIREBALL      = 22;
int VFX_FNF_FIRESTORM     = 23;
int VFX_FNF_IMPLOSION     = 24;
//int VFX_FNF_MASS_HASTE   = 25;
int VFX_FNF_MASS_HEAL     = 26;
int VFX_FNF_MASS_MIND_AFFECTING = 27;
int VFX_FNF_METEOR_SWARM  = 28;
int VFX_FNF_NATURES_BALANCE = 29;
int VFX_FNF_PWKILL        = 30;
int VFX_FNF_PWSTUN        = 31;
int VFX_FNF_SUMMON_GATE   = 32;
int VFX_FNF_SUMMON_MONSTER_1 = 33;
int VFX_FNF_SUMMON_MONSTER_2 = 34;
int VFX_FNF_SUMMON_MONSTER_3 = 35;
int VFX_FNF_SUMMON_UNDEAD = 36;
int VFX_FNF_SUNBEAM       = 37;
int VFX_FNF_TIME_STOP     = 38;
int VFX_FNF_WAIL_O_BANSHEES = 39;
int VFX_FNF_WEIRD         = 40;
int VFX_FNF_WORD          = 41;
int VFX_IMP_AC_BONUS      = 42;
int VFX_IMP_ACID_L        = 43;
int VFX_IMP_ACID_S        = 44;
//int VFX_IMP_ALTER_WEAPON = 45;
int VFX_IMP_BLIND_DEAF_M  = 46;
int VFX_IMP_BREACH        = 47;
int VFX_IMP_CONFUSION_S   = 48;
int VFX_IMP_DAZED_S       = 49;
int VFX_IMP_DEATH         = 50;
int VFX_IMP_DISEASE_S     = 51;
int VFX_IMP_DISPEL        = 52;
int VFX_IMP_DISPEL_DISJUNCTION = 53;
int VFX_IMP_DIVINE_STRIKE_FIRE = 54;
int VFX_IMP_DIVINE_STRIKE_HOLY = 55;
int VFX_IMP_DOMINATE_S    = 56;
int VFX_IMP_DOOM          = 57;
int VFX_IMP_FEAR_S        = 58;
//int VFX_IMP_FLAME_L      = 59;
int VFX_IMP_FLAME_M       = 60;
int VFX_IMP_FLAME_S       = 61;
int VFX_IMP_FROST_L       = 62;
int VFX_IMP_FROST_S       = 63;
int VFX_IMP_GREASE        = 64;

```

```

int VFX_IMP_HASTE = 65 ;
int VFX_IMP_HEALING_G = 66 ;
int VFX_IMP_HEALING_L = 67 ;
int VFX_IMP_HEALING_M = 68 ;
int VFX_IMP_HEALING_S = 69 ;
int VFX_IMP_HEALING_X = 70 ;
int VFX_IMP_HOLY_AID = 71 ;
int VFX_IMP_KNOCK = 72 ;
int VFX_BEAM_LIGHTNING = 73 ;
int VFX_IMP_LIGHTNING_M = 74 ;
int VFX_IMP_LIGHTNING_S = 75 ;
int VFX_IMP_MAGBLUE = 76 ;
//int VFX_IMP_MAGBLUE2 = 77 ;
//int VFX_IMP_MAGBLUE3 = 78 ;
//int VFX_IMP_MAGBLUE4 = 79 ;
//int VFX_IMP_MAGBLUE5 = 80 ;
int VFX_IMP_NEGATIVE_ENERGY = 81 ;
int VFX_DUR_PARALYZE_HOLD = 82 ;
int VFX_IMP_POISON_L = 83 ;
int VFX_IMP_POISON_S = 84 ;
int VFX_IMP_POLYMORPH = 85 ;
int VFX_IMP_PULSE_COLD = 86 ;
int VFX_IMP_PULSE_FIRE = 87 ;
int VFX_IMP_PULSE_HOLY = 88 ;
int VFX_IMP_PULSE_NEGATIVE = 89 ;
int VFX_IMP_RAISE_DEAD = 90 ;
int VFX_IMP_REDUCE_ABILITY_SCORE = 91 ;
int VFX_IMP_REMOVE_CONDITION = 92 ;
int VFX_IMP_SILENCE = 93 ;
int VFX_IMP_SLEEP = 94 ;
int VFX_IMP_SLOW = 95 ;
int VFX_IMP_SONIC = 96 ;
int VFX_IMP_STUN = 97 ;
int VFX_IMP_SUNSTRIKE = 98 ;
int VFX_IMP_UNSUMMON = 99 ;
int VFX_COM_SPECIAL_BLUE_RED = 100 ;
int VFX_COM_SPECIAL_PINK_ORANGE = 101 ;
int VFX_COM_SPECIAL_RED_WHITE = 102 ;
int VFX_COM_SPECIAL_RED_ORANGE = 103 ;
int VFX_COM_SPECIAL_WHITE_BLUE = 104 ;
int VFX_COM_SPECIAL_WHITE_ORANGE = 105 ;
int VFX_COM_BLOOD_REG_WIMP = 106 ;
int VFX_COM_BLOOD_LRG_WIMP = 107 ;
int VFX_COM_BLOOD_CRT_WIMP = 108 ;
int VFX_COM_BLOOD_REG_RED = 109 ;
int VFX_COM_BLOOD_REG_GREEN = 110 ;
int VFX_COM_BLOOD_REG_YELLOW = 111 ;
int VFX_COM_BLOOD_LRG_RED = 112 ;
int VFX_COM_BLOOD_LRG_GREEN = 113 ;
int VFX_COM_BLOOD_LRG_YELLOW = 114 ;
int VFX_COM_BLOOD_CRT_RED = 115 ;
int VFX_COM_BLOOD_CRT_GREEN = 116 ;
int VFX_COM_BLOOD_CRT_YELLOW = 117 ;
int VFX_COM_SPARKS_PARRY = 118 ;
//int VFX_COM_GIB = 119 ;
int VFX_COM_UNLOAD_MODEL = 120 ;
int VFX_COM_CHUNK_RED_SMALL = 121 ;
int VFX_COM_CHUNK_RED_MEDIUM = 122 ;
int VFX_COM_CHUNK_GREEN_SMALL = 123 ;
int VFX_COM_CHUNK_GREEN_MEDIUM = 124 ;
int VFX_COM_CHUNK_YELLOW_SMALL = 125 ;
int VFX_COM_CHUNK_YELLOW_MEDIUM = 126 ;
//int VFX_ITM_ACID = 127 ;
//int VFX_ITM_FIRE = 128 ;
//int VFX_ITM_FROST = 129 ;
//int VFX_ITM_ILLUMINATED_BLUE = 130 ;
//int VFX_ITM_ILLUMINATED_PURPLE = 131 ;
//int VFX_ITM_ILLUMINATED_RED = 132 ;
//int VFX_ITM_LIGHTNING = 133 ;
//int VFX_ITM_PULSING_BLUE = 134 ;
//int VFX_ITM_PULSING_PURPLE = 135 ;

```

```
//int VFX_ITM_PULSING_RED           = 136 ;
//int VFX_ITM_SMOKING              = 137 ;
int VFX_DUR_SPELLTURNING          = 138;
int VFX_IMP_IMPROVE_ABILITY_SCORE = 139;
int VFX_IMP_CHARM                 = 140;
int VFX_IMP_MAGICAL_VISION        = 141;
//int VFX_IMP_LAW_HELP             = 142;
//int VFX_IMP_CHAOS_HELP           = 143;
int VFX_IMP_EVIL_HELP             = 144;
int VFX_IMP_GOOD_HELP            = 145;
int VFX_IMP_DEATH_WARD           = 146;
int VFX_DUR_ELEMENTAL_SHIELD     = 147;
int VFX_DUR_LIGHT                = 148;
int VFX_IMP_MAGIC_PROTECTION     = 149;
int VFX_IMP_SUPER_HEROISM        = 150;
int VFX_FNF_STORM                = 151;
int VFX_IMP_ELEMENTAL_PROTECTION = 152;
int VFX_DUR_LIGHT_BLUE_5         = 153;
int VFX_DUR_LIGHT_BLUE_10        = 154;
int VFX_DUR_LIGHT_BLUE_15        = 155;
int VFX_DUR_LIGHT_BLUE_20        = 156;
int VFX_DUR_LIGHT_YELLOW_5       = 157;
int VFX_DUR_LIGHT_YELLOW_10      = 158;
int VFX_DUR_LIGHT_YELLOW_15      = 159;
int VFX_DUR_LIGHT_YELLOW_20      = 160;
int VFX_DUR_LIGHT_PURPLE_5       = 161;
int VFX_DUR_LIGHT_PURPLE_10      = 162;
int VFX_DUR_LIGHT_PURPLE_15      = 163;
int VFX_DUR_LIGHT_PURPLE_20      = 164;
int VFX_DUR_LIGHT_RED_5          = 165;
int VFX_DUR_LIGHT_RED_10         = 166;
int VFX_DUR_LIGHT_RED_15         = 167;
int VFX_DUR_LIGHT_RED_20         = 168;
int VFX_DUR_LIGHT_ORANGE_5       = 169;
int VFX_DUR_LIGHT_ORANGE_10      = 170;
int VFX_DUR_LIGHT_ORANGE_15      = 171;
int VFX_DUR_LIGHT_ORANGE_20      = 172;
int VFX_DUR_LIGHT_WHITE_5        = 173;
int VFX_DUR_LIGHT_WHITE_10       = 174;
int VFX_DUR_LIGHT_WHITE_15       = 175;
int VFX_DUR_LIGHT_WHITE_20       = 176;
int VFX_DUR_LIGHT_GREY_5         = 177;
int VFX_DUR_LIGHT_GREY_10        = 178;
int VFX_DUR_LIGHT_GREY_15        = 179;
int VFX_DUR_LIGHT_GREY_20        = 180;
int VFX_IMP_MIRV                 = 181;
int VFX_DUR_DARKVISION           = 182;
int VFX_FNF_SOUND_BURST          = 183;
int VFX_FNF_STRIKE_HOLY         = 184;
int VFX_FNF_LOS_EVIL_10         = 185;
int VFX_FNF_LOS_EVIL_20         = 186;
int VFX_FNF_LOS_EVIL_30         = 187;
int VFX_FNF_LOS_HOLY_10         = 188;
int VFX_FNF_LOS_HOLY_20         = 189;
int VFX_FNF_LOS_HOLY_30         = 190;
int VFX_FNF_LOS_NORMAL_10       = 191;
int VFX_FNF_LOS_NORMAL_20       = 192;
int VFX_FNF_LOS_NORMAL_30       = 193;
int VFX_IMP_HEAD_ACID            = 194;
int VFX_IMP_HEAD_FIRE            = 195;
int VFX_IMP_HEAD_SONIC           = 196;
int VFX_IMP_HEAD_ELECTRICITY     = 197;
int VFX_IMP_HEAD_COLD            = 198;
int VFX_IMP_HEAD_HOLY            = 199;
int VFX_IMP_HEAD_NATURE          = 200;
int VFX_IMP_HEAD_HEAL            = 201;
int VFX_IMP_HEAD_MIND            = 202;
int VFX_IMP_HEAD_EVIL            = 203;
int VFX_IMP_HEAD_ODD             = 204;
int VFX_DUR_CESSATE_NEUTRAL      = 205;
int VFX_DUR_CESSATE_POSITIVE     = 206;
```

```
int VFX_DUR_CESSATE_NEGATIVE = 207;
int VFX_DUR_MIND_AFFECTING_DISABLED = 208;
int VFX_DUR_MIND_AFFECTING_DOMINATED = 209;
int VFX_BEAM_FIRE = 210;
int VFX_BEAM_COLD = 211;
int VFX_BEAM_HOLY = 212;
int VFX_BEAM_MIND = 213;
int VFX_BEAM_EVIL = 214;
int VFX_BEAM_ODD = 215;
int VFX_BEAM_FIRE_LASH = 216;
int VFX_IMP_DEATH_L = 217;
int VFX_DUR_MIND_AFFECTING_FEAR = 218;
int VFX_FNF_SUMMON_CELESTIAL = 219;
int VFX_DUR_GLOBE_MINOR = 220;
int VFX_IMP_RESTORATION_LESSER = 221;
int VFX_IMP_RESTORATION = 222;
int VFX_IMP_RESTORATION_GREATER = 223;
int VFX_DUR_PROTECTION_ELEMENTS = 224;
int VFX_DUR_PROTECTION_GOOD_MINOR = 225;
int VFX_DUR_PROTECTION_GOOD_MAJOR = 226;
int VFX_DUR_PROTECTION_EVIL_MINOR = 227;
int VFX_DUR_PROTECTION_EVIL_MAJOR = 228;
int VFX_DUR_MAGICAL_SIGHT = 229;
int VFX_DUR_WEB_MASS = 230;
int VFX_FNF_ICESTORM = 231;
int VFX_DUR_PARALYZED = 232;
int VFX_IMP_MIRV_FLAME = 233;
int VFX_IMP_DESTRUCTION = 234;
int VFX_COM_CHUNK_RED_LARGE = 235;
int VFX_COM_CHUNK_BONE_MEDIUM = 236;
int VFX_COM_BLOOD_SPARK_SMALL = 237;
int VFX_COM_BLOOD_SPARK_MEDIUM = 238;
int VFX_COM_BLOOD_SPARK_LARGE = 239;
int VFX_DUR_GHOSTLY_PULSE = 240;
int VFX_FNF_HORRID_WILTING = 241;
int VFX_DUR_BLINDVISION = 242;
int VFX_DUR_LOWLIGHTVISION = 243;
int VFX_DUR_ULTRAVISION = 244;
int VFX_DUR_MIRV_ACID = 245;
int VFX_IMP_HARM = 246;
int VFX_DUR_BLIND = 247;
int VFX_DUR_ANTI_LIGHT_10 = 248;
int VFX_DUR_MAGIC_RESISTANCE = 249;
int VFX_IMP_MAGIC_RESISTANCE_USE = 250;
int VFX_IMP_GLOBE_USE = 251;
int VFX_IMP_WILL_SAVING_THROW_USE = 252;
int VFX_IMP_SPIKE_TRAP = 253;
int VFX_IMP_SPELL_MANTLE_USE = 254;
int VFX_IMP_FORTITUDE_SAVING_THROW_USE = 255;
int VFX_IMP_REFLEX_SAVE_THROW_USE = 256;
int VFX_FNF_GAS_EXPLOSION_ACID = 257;
int VFX_FNF_GAS_EXPLOSION_EVIL = 258;
int VFX_FNF_GAS_EXPLOSION_NATURE = 259;
int VFX_FNF_GAS_EXPLOSION_FIRE = 260;
int VFX_FNF_GAS_EXPLOSION_GREASE = 261;
int VFX_FNF_GAS_EXPLOSION_MIND = 262;
int VFX_FNF_SMOKE_PUFF = 263;
int VFX_IMP_PULSE_WATER = 264;
int VFX_IMP_PULSE_WIND = 265;
int VFX_IMP_PULSE_NATURE = 266;
int VFX_DUR_AURA_COLD = 267;
int VFX_DUR_AURA_FIRE = 268;
int VFX_DUR_AURA_POISON = 269;
int VFX_DUR_AURA_DISEASE = 270;
int VFX_DUR_AURA_ODD = 271;
int VFX_DUR_AURA_SILENCE = 272;
int VFX_IMP_AURA_HOLY = 273;
int VFX_IMP_AURA_UNEARTHLY = 274;
int VFX_IMP_AURA_FEAR = 275;
int VFX_IMP_AURA_NEGATIVE_ENERGY = 276;
int VFX_DUR_BARD_SONG = 277;
```

```
int VFX_FNF_HOWL_MIND           = 278;
int VFX_FNF_HOWL_ODD           = 279;
int VFX_COM_HIT_FIRE           = 280;
int VFX_COM_HIT_FROST          = 281;
int VFX_COM_HIT_ELECTRICAL     = 282;
int VFX_COM_HIT_ACID           = 283;
int VFX_COM_HIT_SONIC          = 284;
int VFX_FNF_HOWL_WAR_CRY       = 285;
int VFX_FNF_SCREEN_SHAKE       = 286;
int VFX_FNF_SCREEN_BUMP        = 287;
int VFX_COM_HIT_NEGATIVE       = 288;
int VFX_COM_HIT_DIVINE         = 289;
int VFX_FNF_HOWL_WAR_CRY_FEMALE = 290;
int VFX_DUR_AURA_DRAGON_FEAR   = 291;
```

Appendix F – Area Effect Constants

```
int AOE_PER_FOGACID           = 0;
int AOE_PER_FOGFIRE          = 1;
int AOE_PER_FOGSTINK         = 2;
int AOE_PER_FOGKILL          = 3;
int AOE_PER_FOGMIND          = 4;
int AOE_PER_WALLFIRE         = 5;
int AOE_PER_WALLWIND         = 6;
int AOE_PER_WALLBLADE        = 7;
int AOE_PER_WEB               = 8;
int AOE_PER_ENTANGLE         = 9;
//int AOE_PER_CHAOS           = 10;
int AOE_PER_DARKNESS         = 11;
int AOE_MOB_CIRCEVIL         = 12;
int AOE_MOB_CIRCGOOD         = 13;
int AOE_MOB_CIRCCLAW         = 14;
int AOE_MOB_CIRCCHAOS        = 15;
int AOE_MOB_FEAR              = 16;
int AOE_MOB_BLINDING         = 17;
int AOE_MOB_UNEARTHLY        = 18;
int AOE_MOB_MENACE           = 19;
//int AOE_MOB_UNNATURAL       = 20;
int AOE_MOB_STUN              = 21;
int AOE_MOB_PROTECTION        = 22;
int AOE_MOB_FIRE              = 23;
int AOE_MOB_FROST             = 24;
int AOE_MOB_ELECTRICAL       = 25;
int AOE_PER_FOGGHOUL         = 26;
int AOE_MOB_TYRANT_FOG       = 27;
int AOE_PER_STORM             = 28;
int AOE_PER_INVIS_SPHERE     = 29;
int AOE_MOB_SILENCE           = 30;
int AOE_PER_DELAY_BLAST_FIREBALL = 31;
int AOE_PER_GREASE            = 32;
int AOE_PER_CREEPING_DOOM     = 33;
int AOE_PER_EVARDS_BLACK_TENTACLES = 34;
int AOE_MOB_INVISIBILITY_PURGE = 35;
int AOE_MOB_DRAGON_FEAR      = 36;
```

Appendix G – Cyrus' & DanieleB's Loot Tables

```

//
// Spawn Loot
//
// --- begin added for cyrus loot tables -----
#include "gen_gold"
#include "gen_gems"
#include "gen_magic"
// --- end added for cyrus loot tables -----
int ParseFlagValue(string sName, string sFlag, int nDigits, int nDefault);
int ParseSubFlagValue(string sName, string sFlag, int nDigits, string sSubFlag, int
nSubDigits, int nDefault);
object GetChildByTag(object oSpawn, string sChildTag);
object GetChildByNumber(object oSpawn, int nChildNum);
void DeactivateSpawn(object oSpawn);
void DeactivateSpawnsByTag(string sSpawnTag);
void DeactivateAllSpawns();
void DespawnChildren(object oSpawn);
void DespawnChildrenByTag(object oSpawn, string sSpawnTag);
//
//
void LootTable(object oSpawn, object oSpawned, int nLootTable)
{
    // Initialize
    object oItem;
    string sTemplate;
    int nStack;

//
// Only Make Modifications Between These Lines
// -----
/*
//:////////////////////
//: Name Cyrus's Loot Table's
//: FileName spawn_cfg_loot
//: Copyright (c) 2001 Bioware Corp.
//:////////////////////
NOTE: this is Neshke's Extended Spawn System
all i did was make a custom Loot table system
the follows the DMG guide as closely as i could
EXAMPLES on how to modify the tables if you choose to

GenerateGold(oSpawned, 40, 1000, 6000);

40 percent chance for the table to drop gold
1000 minimum gold, 6000 max gold

GenerateGems(oSpawned, 60, 10, 40);

60 percent chance for the table to drop gems
10 gems minimum, 40 gems max

GenerateMagic(oSpawned, 30, 3);

30 percent chance to drop magical items
3 rolls on the gen_magic chart(s)
*/
//:////////////////////
//: Created By: ONLY the LOOT TABLE CODE : Cyrus_1469
//: Created On: 9/20/2002
//:////////////////////

// Table 00
if (nLootTable == 0)
{
    // 50% Chance
    if (d100(1) > 50)

```

```

    {
        // Created Custom Item with ResRef of magic sword
        sTemplate = "magic sword";
        nStack = 1;
    }
    oItem = CreateItemOnObject(sTemplate, oSpawned, nStack);
}
//

// Calling Loot Table 1
if (nLootTable == 1)
{
    GenerateGold(oSpawned, 80, 10, 100);
    GenerateGems(oSpawned, 0, 1, 4);
    GenerateMagic(oSpawned, 0, 3);
}
// Calling Loot Table 2
if (nLootTable == 2)
{
    GenerateGold(oSpawned, 100, 100, 500);
    GenerateGems(oSpawned, 100, 1, 2);
    GenerateMagic(oSpawned, 10, 1);
}
// Calling Loot Table 3
if (nLootTable == 3)
{
    GenerateGems(oSpawned, 25, 1, 6);
    GenerateMagic(oSpawned, 10, 2);
}
// Calling Loot Table 4
if (nLootTable == 4)
{
    GenerateGold(oSpawned, 50, 1000, 3000);
    GenerateGems(oSpawned, 30, 1, 10);
    GenerateMagic(oSpawned, 15, 2);
}
// Calling Loot Table 5
if (nLootTable == 5)
{
    GenerateGold(oSpawned, 25, 1000, 4000);
    GenerateGems(oSpawned, 15, 1, 12);
    GenerateMagic(oSpawned, 25, 3);
}
// Calling Loot Table 6
if (nLootTable == 6)
{
    GenerateGold(oSpawned, 40, 1000, 6000);
    GenerateGems(oSpawned, 20, 2, 20);
    GenerateMagic(oSpawned, 30, 5);
}
// Calling Loot Table 7
if (nLootTable == 7)
{
    GenerateGems(oSpawned, 100, 2000, 20000);
    GenerateMagic(oSpawned, 35, 5);
}
// Calling Loot Table 8
if (nLootTable == 8)
{
    GenerateGold(oSpawned, 55, 2000, 20000);
    GenerateGems(oSpawned, 50, 3, 30);
    GenerateMagic(oSpawned, 15, 6);
}
// Calling Loot Table 9
if (nLootTable == 9)
{
    GenerateGems(oSpawned, 55, 2, 12);
    GenerateMagic(oSpawned, 15, 1);
}
// Calling Loot Table 10
if (nLootTable == 10)

```



```

        {
            GenerateGold(oSpawned, 100, 10, 100);
        }
//Calling Loot Table 11
if (nLootTable == 11)
    {
        GenerateGems(oSpawned, 100, 1, 4);
    }
// Calling Loot Table 12
if (nLootTable == 12)
    {
        GenerateGold(oSpawned, 100, 2, 20);
        GenerateGems(oSpawned, 100, 2, 8);
    }
//Calling Loot Table 13
if (nLootTable == 13)
    {
        GenerateMagic(oSpawned, 100, 1);
    }
// Calling Loot Table 14
if (nLootTable == 14)
    {
        GenerateMagic(oSpawned, 100, 1);
    }
// Calling Loot Table 15
if (nLootTable == 15)
    {
        GenerateGems(oSpawned, 90, 2, 16);
        GenerateMagic(oSpawned, 70, 1);
    }
// Calling Loot Table 16
if (nLootTable == 16)
    {
        GenerateMagic(oSpawned, 60, 2);
    }
// Calling Loot Table 17
if (nLootTable == 17)
    {
        GenerateGold(oSpawned, 100, 5, 30);
        GenerateGems(oSpawned, 60, 2, 16);
        GenerateMagic(oSpawned, 60, 2);
    }
// Calling Loot Table 18
if (nLootTable == 18)
    {
        GenerateMagic(oSpawned, 100, 2);
    }
// Calling Loot Table 19
if (nLootTable == 19)
    {
        GenerateGold(oSpawned, 100, 200, 1200);
    }
// Calling Loot Table 20
if (nLootTable == 20)
    {
        GenerateGold(oSpawned, 100, 100, 600);
        GenerateGems(oSpawned, 55, 1, 6);
        GenerateMagic(oSpawned, 50, 3);
    }

// Merchant-based loot - from DanieleB NESS scripts
if( nLootTable >= 500 )
{
    object oStore = OBJECT_INVALID;
    oStore = GetObjectByTag( "LOOT_" + IntToString(nLootTable));

    object oItem;
    int nCount;
    int nAmount;
    if( GetIsObjectValid( oStore ) )

```

```

{
// -- check if we already know item count
nCount = GetLocalInt( oStore , "nItemCount" );
if( nCount <= 0 )
{
// -- Count Items in Store Inventory
oItem = GetFirstItemInInventory( oStore );
while( GetIsObjectValid( oItem ) )
{
nCount++;
oItem = GetNextItemInInventory( oStore );
}
SetLocalInt( oStore , "nItemCount" , nCount );
}
// -- probability for multiple items
nAmount = d100();

// Cereborn: removed 12/31/02
// Old:
// int nProbOneItem    = 50;    // 50% chance 1 item
// int nProbTwoItems   = 15;    // 15% chance 2 items
// int nProbThreeItems = 5;     // 5%  chance 3 items
//                                     // 30% chance no items ( implied )
// New:
int nProbOneItem    = GetLocalInt(oSpawn, "f_LootTable1ItemChance");
int nProbTwoItems   = GetLocalInt(oSpawn, "f_LootTable2ItemChance");
int nProbThreeItems = GetLocalInt(oSpawn, "f_LootTable3ItemChance");

if( nAmount <= nProbThreeItems )
    nAmount = 3;
else
if( nAmount <= nProbThreeItems + nProbTwoItems )
    nAmount = 2;
else
if( nAmount <= nProbThreeItems + nProbTwoItems + nProbOneItem )
    nAmount = 1;
else
    nAmount = 0;
// -- Generate nAmount items on oSpawned
while( nAmount > 0 )
{
// -- Determine random item
int nSelected;
int nRand = Random( nCount ) + 1;
// -- Get the item
oItem = GetFirstItemInInventory( oStore );

for( nSelected = 1 ; nSelected < nRand ; nSelected++ )
{
oItem = GetNextItemInInventory( oStore );
}
// -- Grab item template
if (oItem != OBJECT_INVALID)
{
sTemplate = GetResRef( oItem );
}

// -- Checks to see if this it is a ammo or thrown item and creates more
in the stack
string sRoot = GetStringLowerCase( GetSubString( sTemplate , 0 , 6 ) );
if( sRoot == "nw_wam" || sRoot == "nw_wth" )
{
nStack = Random( 30 ) + 1;
}
else
// -- Check if the item is Gold, and creates more in Stack
//    small amount generated : gold placement should maybe be handled in
some other way.
if( GetStringLowerCase( sTemplate ) == "nw_it_gold001" )
    nStack = Random( 30 ) + 5;
else

```

```
        if( nStack < 1 )
            nStack = 1;
        // -- create the item on oSpawned
        oItem = CreateItemOnObject( sTemplate , oSpawned , nStack );
        // -- decrement the Item Amount counter
        nAmount--;
    }
else
{
    // Write to log
    PrintString( "Could not find Loot Merchant [" +
        "LOOT_" +
        IntToString( nLootTable ) +
        "]" for Spawn Waypoint : " +
        GetName( oSpawn ) );
}

// -----
// Only Make Modifications Between These Lines
//
}
```