

# NWN2 Toolset Guide

## Volume I

### Edition 6.1

This work is intended to serve as a basic user's guide and reference for the Neverwinter Nights 2 module-building toolset, hereafter denoted by NWN2. Volume II includes a set of supplementary appendices, volume III can be used as a scripting reference, while volume IV has some example scripts.

As the documentation for the toolset is somewhat limited, I began developing this work as a personal reference during the process of gaining familiarity with it's capabilities. The information was assembled by reading the tool's built-in help page, experimenting with the various toolset features, perusing the game's install directory, and gleaning knowledge from the internet – especially from the very useful Bioware NWN2 forums. For a list of additional

resources used, see the references section at the end of the document.

In cases where I was unable to determine the purpose of a feature or how to resolve a problem, I noted these in italics. If you find any corrections or have suggestions, edits and additions that would improve this document, I would enjoy hearing from you. Please send me a friendly e-mail.

Other than being a licensed purchaser of the software, I have no affiliation with the firms that build and maintain this toolset. Thank you for taking the time to look through this work, and I hope you find it of some use.

—[Bob Hall](#)

September 1, 2013

## Table of Contents

<a href="#">Interface</a> .....	2	<a href="#">Doors</a> .....	58	<a href="#">Editing</a> .....	97
<a href="#">Menus</a> .....	2	<a href="#">Stores</a> .....	63	<a href="#">Tabs</a> .....	99
<a href="#">Toolbars</a> .....	5	<a href="#">Placeables</a> .....	67	<a href="#">Tools</a> .....	104
<a href="#">Panels</a> .....	7	<a href="#">Triggers</a> .....	76	<a href="#">Journal</a> .....	104
<a href="#">Patching</a> .....	8	<a href="#">Encounters</a> .....	78	<a href="#">World Map Editor</a> .....	105
<a href="#">Creating a Game</a> .....	10	<a href="#">Sounds</a> .....	80	<a href="#">Overland Map</a> .....	106
<a href="#">Modules</a> .....	13	<a href="#">Waypoints</a> .....	83	<a href="#">Plugins</a> .....	107
<a href="#">Areas</a> .....	16	<a href="#">Static Cameras</a> .....	85	<a href="#">Writing Scripts</a> .....	111
<a href="#">Editing</a> .....	16	<a href="#">Lights</a> .....	86	<a href="#">Editing</a> .....	111
<a href="#">Exterior Areas</a> .....	21	<a href="#">Trees</a> .....	89	<a href="#">Debugging</a> .....	115
<a href="#">Interior Areas</a> .....	32	<a href="#">Placed Effects</a> .....	91	<a href="#">Examples</a> .....	118
<a href="#">Blueprints</a> .....	37	<a href="#">Prefabs</a> .....	94	<a href="#">Item Scripts</a> .....	122
<a href="#">Items</a> .....	38	<a href="#">Conversations</a> .....	95	<a href="#">References</a> .....	124
<a href="#">Creatures</a> .....	47			<a href="#">Revision History</a> .....	125

## Interface

The NWN2 Toolset is a graphical tool for generating a computer role-playing adventure that can be played using the NWN2 game engine. It is installed as a standard part of the game package and can be launched from the game startup interface.<sup>1</sup>

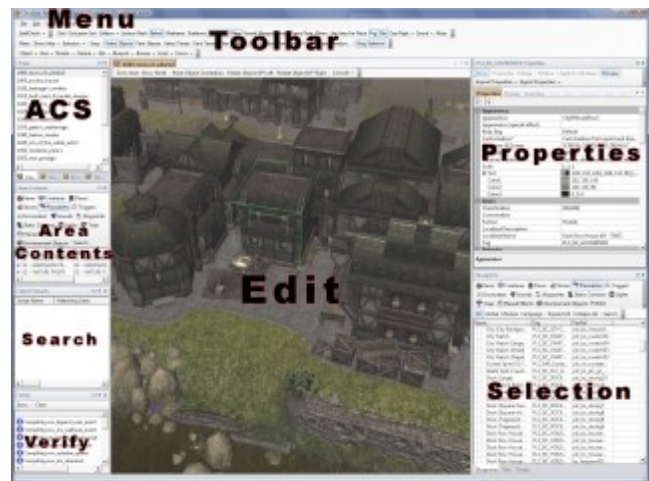
The game list Windows XP as the requirement operating system. When run on a Vista system, the toolset will immediately stop and generated an error dialog. To get the toolset to run in Vista, bring up the application's Properties window from the desktop, select the Compatibility tab and configure it to run in compatibility mode for "Windows XP (Service Pack 2)".<sup>2</sup> (It is also recommended that you also set the "Disable visual themes" option.) After clicking Okay, the toolset should launch properly. If it still does not, you may have an outdated version of DirectX loaded on your PC. You should have version 9.0c or subsequent.

It can take about twenty seconds for the toolset to finish loading into memory. Once the toolset is running, you will be presented with an editing interface consisting of [menus](#), [toolbars](#) and [panels](#). These components are described in more detail below.

The following terminology is used in the interface:

- A [module](#) is an adventure that can be played through the game engine. A campaign consists of two or more modules.
- An [area](#) represents a rectangular region of [exterior](#) or [interior](#) terrain that provides a playing surface.
- An object is a set of data representing a component or control record within the game. [Blueprints](#) are object definitions that represent [creatures](#), [items](#), [doors](#), [buildings](#), [lights](#), [trees](#), furniture, and so forth.

- A [script](#) is sets of commands that are used to control objects and manage the game environment.
- A [conversation](#) is a branching tree of text strings that are used simulate verbal interactions.
- A [journal](#) is a series of text descriptions that are used to provide module status information to the player.



*The toolkit interface*

## Menus

The menu bar at the top of the window provides various options for managing the toolset. It consists of the File, Edit, View, Plugins and Help menus. The menus and menu items are configured with various accelerator key options listed in brackets. In the descriptions below, a notation such as 'alt-x+y' means to hold down the 'alt' key, then press the 'x' and 'y' keys in succession.

## File menu

The File menu [alt-f] combines two different functions. The first is to provide access to the files where the toolset edits will be saved. The menu also includes several options for preparing an edited module for use with the game engine.

- *New* [alt-f+n] has a submenu for creating a new module, area, conversation or script. These are the

<sup>1</sup> It can also be launched from the PC's Start menu under the Atari pick. By default the game installer did not create a toolset icon on my Vista desktop. If you locate the application folder, you can create a desktop icon of the menu toolset by a right-click and selecting 'Create Startup Icon'.

<sup>2</sup> If you are reloading the game, the OS may have cached the compatibility settings from before. In this case, try changing it to a lower compatibility setting then change it back.

## Menus

components that are used to create a game.

- The *Open...* [alt-f+o or ctrl-o] and *Open Directory...* [alt-f+d] items will allow you to browse for a saved module to edit. By default your modules are saved to the “Neverwinter Nights 2” folder under your personal documents. You can also open the main campaign modules from the 'Modules' under the game's install directory. (Be careful not to save changes to these modules or the game might not run correctly.) For a sample module, try opening '0100\_UninvitedGuests' from the 'Modules' folder.
- The *Save...* [alt-f+s or ctrl-f+s], *Save As...* [alt-f+a] and *Save Directory...* [alt-f+v] options save your current edits to either a file with a '.mod' extension or to a directory tree.

On some occasions, module files have been known to become corrupted and unrecoverable. (I've run into this twice thus far.) When this happens, the module will open but there will be no content remaining from your prior edits. If this happens then you will need to restore the module from a backup.

Some have suggested that saving your work as a directory, rather than a module file, will make it more robust against corruption. Turning off the auto-save feature also seems to reduce the frequency of module corruption. (See the View menu below.) Alternatively, you can use the Export menu item to save copies of the Areas, Conversations, Scripts and other significant blueprints. The latter operation has saved me re-work on more than one occasion.

- *Bake Current Area* [alt-f+b or ctrl-B] and *Bake All Areas* [ctrl-A] triggers an automated routine that defines the regions of an area map that can be traversed by the player's characters. It is an integral part of the process for preparing an area for use in a module, and must always be run at least once per area before playing the game. Failure to bake an area can make it impassible and could even crash the game.
- *Make Client Pack* (PWC File) is used to create a reduced size file that can be copied to client systems. It allows participants to join a multi-player game using

a module built with the toolset.

- *Open Conversation/Script* [ctrl-O] opens a dialog interface that can be used to open the various bundled scripts included with the toolset. It will open the selected file in the Edit panel (described in the chapter on [writing scripts](#)). Along the top of the dialog is a filter text box that can be used for look for specific string patterns in the files. (For example, enter 'gc\_' for the [conversation](#) condition scripts.)
- *Import...* [alt-f+i] can be used to load Encapsulated Resource Files (.erf) into your current edit session, while *Export...* [alt-f+e] will save specific module components to '.erf' files. This is a convenient method for transferring unique components between different modules, or archiving major edits. See also the ERF Editor pick under the Plug-ins menu.
- The *Run Module* [ctrl-f5] pick will launch a trial version of your current module at the site of your Start Location. It automatically assigns you the PC that appears first in the list of pre-generated characters, which are ordered alphabetically by first name.<sup>3</sup> If you do not have the game disk installed, a dialog will appear asking you to insert the CD/DVD.
- *Verify Module...* [shift-f5] will perform certain validation tests on your current module. The results are displayed in a Verify panel.
- *Compile* will perform a compilation of scripts used in your module. The results are displayed in the Verify panel. If you use custom include files in your scripts, it is important to recompile all scripts in your module each time you modify an include file.
- *Close Tab* [alt-f+c] will close the currently selected

---

<sup>3</sup> If you don't want to test your module using the first stock character in the list, you can use the NWN2 game to create and export a customized character that has a suitable name. (For this purpose I created a NWN2 character with a name that begins with “Aa”, such as Aaron or Aadela.)

The toolset will always select the first '.bic' file in the 'Neverwinter Nights 2/localvault' folder in the documents area of your home folder. To use a pre-existing character, insert a numeral at the start of the corresponding '.bic' file name. This will put the file at the start of the alphabetical sort order.

## Menus

tab in the Edit panel. Alternatively, you can right-click on an Edit panel tab and select Close from the pop-up menu.

- *Exit* [alt-f+x] will prompt you to save the current edits then quit the toolset.

### Edit menu

These menu items are used for area editing operations.

- Undo [ctrl-z]
- Redo [ctrl-y]
- Cut [ctrl-x]
- Copy [ctrl-c]
- Paste [ctrl-v]

The edit functions can prove unreliable when placing data on the clipboard for transfer between the Toolset and other applications. For example, you may need to make several copy attempts before a selection will be added to the copy buffer.

### View menu

- *Module Properties* [alt-v+p] will display the properties of the currently open module in the Properties panel. This is useful for applying global properties and scripts to a module.
- *Journal* [alt-v+j] will allow you to create and edit journal entries for the module or campaign. See the [Journal](#) section for more details.
- *Factions* [alt-v+f] is a table that shows the likes and dislikes between creatures belonging to different factions. A faction is comparable to an alliance, in which members of the same faction support each other during combat. Likewise, members of a faction that are not directly involved in a conflict will avoid the ongoing combat.
- *2DA File* [alt-v+2] opens a dialog interface that allows you to select one of the two-dimensional data files for viewing.<sup>4</sup>

---

<sup>4</sup> Much of the static game information is configured through these arrays, so they should be referenced when the existing

- *Challenge Rating Editor* allows you to set the challenge rating for creatures that have been created for your module.
- *Mode* selects the type of action performed in the Edit panel. These functions are repeated in the toolbar, so there is little reason to use the menu. Four of the items have equivalent f-keys.
- *Options* [alt-v+o] opens a dialog that allows you to modify the toolset appearance and behavior. Of particular interest:
  - i. *General/Autosave* sets the frequency that automatic saves will be performed (in minutes per save), and how many copies to keep. I turn this off to avoid corruption problems.
  - ii. *Graphics/FarPlane* sets the distance of the far plane while drawing Exterior terrain. This is the blue plane that appears when you scroll well back from the area.
  - iii. *Script/Font* sets the font used for editing scripts.

### Plugins menu

- Process [alt-p+p]
- *Animation Viewer* allows you to view the various animations associated with the different creature appearance types. First, select New Appearance from the File menu, then pick an appearance and click okay. Next, choose a stance, select an animation<sup>5</sup> and add to the queue. Click the green arrow button to loop the animation.
- *Campaign Editor* is used to create campaigns consisting of multiple modules. Opening a campaign built with a patched toolset may result in a "Could not load module" error.
- Community
- *ERF Editor* can be used to view exported resource

toolkit information does not provide sufficient details. See the Arrays section of the second volume for more information.

<sup>5</sup> The stances include UNA = unarmed; 1HS = one-handed weapon; 1HSS = one-handed weapon with shield; DHS = dual-handed weapons; C2H = two-handed club; O2HT = two-handed thrust; O2HS = two-handed swing, and so forth.

## Menus

files created with the *Export...* menu item.

- Model Viewer
- Sound Set Editor
- Universal Blueprint Changer
- *Visual Effects Editor* [alt-p+v] is a dialog interface that can be used to edit '.sef' and '.bbx' files
- *World Map Editor* is a dialog that is used to edit game world map (.WMP) files. See the [World Map Editor](#) section for more details.

Additional items will appear in this menu as new [plug-ins](#) are installed.

### Help menu

- *Help* [alt-h+h] opens up the basic help web page. This is a good place to look for information on how to get started with the toolset. It also includes several examples and screen shots.
- *About* [alt-h+a] lists the current toolset version.

### Toolbars

The default editor has two standard toolbars located just underneath the window menus. The first toolbar allows you to toggle various graphical and acoustical states of the main Edit pane (at center) while it is being used to edit [areas](#). Some of the toolbar effects may vary slightly depending on whether an [exterior](#) or [interior area](#) is being displayed, but in general they are consistent.

- *Grid* shows a coordinate grid across the map when an area is open for editing. On an exterior area, major grid lines are black while the minor lines are red. On indoor area, the grid lines are green and they only appear in open areas.
- *Occlusion Grid* displays the maximum area where the character can roam. On the exterior area maps, this is a rectangle located two major grid lines in from the edge of the map. It shows as red where it intersects with ground that is at or above elevation 0.0. On interior area maps this gives a tiling grid, consisting of four minor lines per major line.

- *Collision* draws boxes around objects showing the collision volumes; presumably where the game engine will check for collisions between objects.
- *Surface Mesh* shows the walkable areas of the map using yellow lines as borders.
- *Baked* (with *Surface Mesh* on) displays the blocked areas of the map after baking the map. After baking it is importance to check the map for impassible grid squares. See the [areas chapter](#) for more details on addressing this problem. Note that activating this option will hide the Surface Mesh on the outer border region of an exterior map.
- *Wireframe* turns off rendering of the terrain and only displays the triangular wireframe that defines the surface altitude.
- *Skeletons* prints a summary line for each non-static [placeable](#), and displays skeleton animation data. The last value in the yellow-hued strings gives a range from the viewer's perspective.
- *Shadows* turns on rendering of shadows from objects but not from the terrain.
- *Water* turns on the water planes. Turning this off is useful for drawing the underwater surface.
- *Normal Mapped Terrain* is (I think) how the terrain will be rendered in the game, including surface bump shadows.
- *Frame Rate* prints a table of video memory information.
- *Bloom* makes brighter patches of the area give off a faint surrounding glow. The net effect can be to make the image more fuzzy and obscure details.<sup>6</sup>
- *Use Area Far Plane* puts a plane through the image perpendicular to the line of sight. The distance to this plane is set via the Far Plane field in the Graphics Options. To set this, choose 'Options...' from the View menu. For exiting purposes, I usually set it to 1000 so it is out of the way.

---

<sup>6</sup> If certain rendering features don't seem to be rendering properly in the toolset, such as the bloom, try exiting and then running the game separately. This resets the rendering engine, so the features may start working again.



## Toolbars

- *Fog* turns on the rendering of the fog settings from the area properties Day/Night Cycle Stages parameters, based upon the current selection from the Day/Night menu (described next).
- *Sky* shows the appearance of the sky ceiling, which is visible when you pan the view all the way down until the scene is horizontal. The appearance is based upon the area properties Day/Night Cycle Stages parameters, which uses the current selection from the Day/Night menu (described next).
- *Day/Night* determines the appearance based on the corresponding Day/Night Cycles Stages configuration. Try different time picks to see how the map will appear under various viewing conditions. Normally you would want to edit an area with this menu set to Daytime or Default. Setting 'Run' and 'Fast' will cycle through the full daytime cycle within the span of half a minute.
- *Sound* controls whether you can hear the Ambient and Placed sounds that will be used during the game. The Placed sounds will vary depending on the Sound chosen and the position of the view. *I.e.* You can move the area view about to see how a place sound will be heard.
- *Music* plays the music selected in the area properties.

The lower toolbar includes display filters and the main editor controls.

- *Filters* is a menu that allows you to fine tune the area's display and selection of objects by their blueprint category. In an area that is crowded with objects, this can be used to select specific objects by hiding the other object categories or making them non-selectable. (You can also select individual objects from the Area Contents panel, although that may require more trial and error.) Note that the hidden objects are not deleted; they are merely concealed from sight.
- *Snap* will cause a door to attach itself to a doorway. On interior areas, this setting will also cause placeables to attach themselves to hookpoints along walls. Hookpoints are the multi-colored, three-axis shapes that appear along walls. If you have trouble assigning a placeable to the right hookpoint, try toggling the snap off, moving the placeable about, then re-enabling snap.
- *Select Objects* [f2] will allow you to pick an object for relocation. Use shift-click to select multiple objects, or drag the cursor in a rectangle to choose the objects in an area. Clicking on a selected object with the shift key down will deselect that object. Clicking and dragging selected objects will move them about the x-y plane. Holding the shift key while pressing PgUp or PgDn will change the object's vertical position. Turning the scrolling knob on the mouse will change the vertical position of the selected objects in larger increments (0.24 per click).
- *Paint Objects* [f3] will cause the currently selected blueprint to appear at the location of the cursor, while remaining at the ground level of the surface. Left-clicking will place a copy of the object at the current position. Press Esc to switch to *Select Objects*.
- *Select Terrain* [f4] will select grid boxes on the current map. Right clicking will switch to Paint Terrain.
- *Paint Terrain* [f5] sets the cursor to use whatever Terrain is currently selected from the Selection panel (see the chapter on [Areas](#)). It is used on exterior areas.
- *Tiles* is used to paint tiles on an interior area using the currently highlighted Tile in the Selection panel. It is also used to select one or more individual tiles to fine tune their properties, such as the colors and textures.
- *Set Start Location* will paint a special object consisting of an red arrow inside a circle. This defines the location where the character will first appear when the module is launched, and the direction [s]he will be facing. If you haven't defined this yet, you will get a query each time the module autosaves.
- *Paint Spawn Point* is used to define the spawn points for the currently selected Trigger object. (See the section on [Triggers](#).)
- *Create Transition...*
- *Drag Selection* turns on a draggable rectangle that can be used to select multiple objects in an area.

## Panels

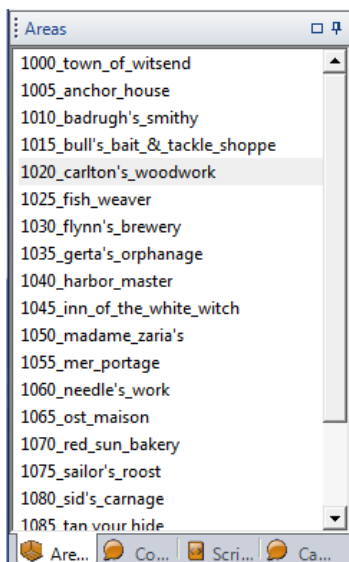
### Panels

There are six standard panels available with the toolset. By default, the main editing panel is located in the center. There are three smaller panels to the left and two along the right. You can move these panels around by dragging their title bar, or manipulate their visibility with the tiny rectangle and pin icons in the upper right of the panel. The tiny rectangle changes the proportionate size of the panel. You can also adjust the panel size by moving the cursor to the panel edges until it changes to a double arrow, then click-dragging. Clicking on the pin icon converts the panel to a tabbed bar along the side of the window. Click on these tabs to display the panel, then click on the pin to restore it to visibility.

As I don't know what titles the game vendor uses to identify the toolset panels, I will use the names below in the remainder of the document.

#### A/C/S Panel

Once a module has been created, the upper-left panel displays the areas, conversations and scripts: basically all of the module features that can be generated by the New submenu item under File. Right clicking in this panel will display a menu of options, such as 'Add' for creating a new addition of the current type. You can switch between the content using the tabs along the bottom of the panel.



Area/Conversation/Scripts panel with the Areas tab selected

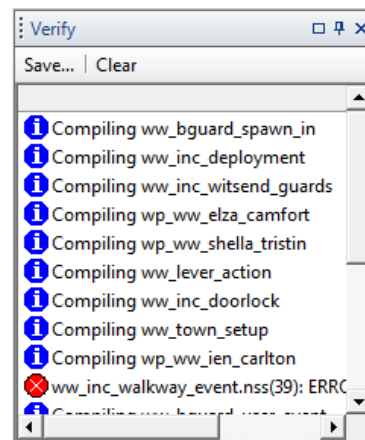
By default this panel will also display Campaign Conversations and Campaign Scripts. When you click on either of these panels, an 'X' icon will appear in the upper right corner. If you click on the 'X', the tab will be removed from the panel. Note, however, I have not found any means to restore these tabs once they are removed.

#### Area Contents Panel

When an area is open in the edit panel, selecting any of the tabs in this panel will display a list of the Placeable objects of the specific type. This can be useful for finding specific objects that you want to edit.<sup>7</sup> Double-clicking on an object in any of the lists will cause the area Edit panel to reposition with the view centered on that item.

The search tab will open a dialog window that can be used to search for a name under one of the tabs. Thus it can be used to search for a "Bridge" under the Placeables listing. If it gets a match, it will select that object. If you select an object in the contents list you can right click and display the properties in a new window. (The 'Graph...' option doesn't seem to do anything.) The Creatures list can also show the inventory and equipped items of the listed creatures.

#### Verify Panel



The Verify panel after compiling all scripts in the module.

A script has an error as indicated by the red octagon icon.

When you execute a Compile operation from the File menu, the results will appear in this list at the bottom left. If you have an extensive set of scripts, it may take many

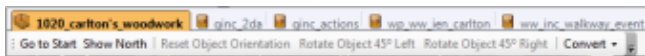
<sup>7</sup> Note that placeables that are converted to environmental objects will be relocated to the Environmental Objects tab.

## Panels

seconds for the list to appear. When it does, you'll have to scroll down to see if any errors occurred.

### Edit Panel

By default, the largest panel lies in the center of the toolset display. This panel is used to edit the entries listed in the A/C/S panel, and to view the built-in scripts and 2DA arrays. To edit an area, click on the Area tab in the A/C/S panel, select the area from the list, then right click and choose "Open". The area will be opened under a tab located at the top of this panel.



*Tabs at the top of the Edit panel*

When multiple items are open, they will each have their own tabs. (There is a limit of three open Areas at a time.) Selecting a tab will bring the associated edit entry to the fore. The tabs can be re-arranged by dragging them horizontally. Entries can be closed by right-clicking over the tab and choosing "Close". You will be prompted to Save your edits.

### Properties Panel

Located at the upper right, this shows the various configuration parameters for the currently selected item, whether it is an object in the area, a blueprint, or an entry in the A/C/S panel. This is used for setting the properties of Module, Areas, Conversations, Blueprints, objects and Tiles.

### Selection Panel

At the lower right is a grab-bag tool for selecting various editing graphics or game objects. This is the primary panel for managing what types of changes you are going to make to an area. The tabs along the bottom of this panel are used to select the general categories, then the corresponding selections and options appear above the tab. The Blueprints tab provides a selection of objects that can be placed in an area. Tiles are used to choose the appearance of grid squares on an interior map, while Terrain is used for controlling how the terrain will be modified on an exterior map.

## Patching

NWN2 patches are available that will fix various issues with the toolset and implement new functionality. If you have internet access, these can be loaded via the game's update control. The vendor recommends that you use automatic patching to update your game and toolset, unless this method fails consistently or you don't have a direct internet connection. It is generally a good idea to keep your game patched to the latest level.

**Warning:** Modifying any of the content under the game's install directory will cause patching to fail. You should restrict your modifications to your Documents folder in your home directory.

To manually update the toolset from version 1.0.1115.0 to version 1.0.1588.0, I pulled version 2.25 of the NWNPatcher tool from the NW Vault site and downloaded the following patches from the site's language patch page:

- nwn2\_pcx1\_english\_from1101115\_to1101116.zip
- nwn2\_pcx1\_english\_from1101116\_to1111152.zip
- nwn2\_pcx1\_english\_from1111152\_to1111153.zip
- nwn2\_pcx1\_english\_from1111153\_to1121295.zip
- nwn2\_pcx1\_english\_from1121295\_to1131407.zip
- nwn2\_pcx1\_english\_from1131407\_to1131409.zip
- nwn2\_pcx1\_english\_from1131409\_to1211549.zip
- nwn2\_pcx1\_english\_from1211549\_to1221586.zip<sup>8</sup>
- nwn2\_pcx1\_english\_from1221586\_to1221587.zip
- nwn2\_pcx1\_english\_from1221587\_to1221588.zip

The nwn2\_pcx1\_english\* patches are applicable to the game with the 'Mask of the Betrayer' expansion.<sup>9</sup> Each patch file is at least 47 Mb, and some are up to 90 Mb in size. Each '.zip'-extension file contains an RTP file, which I extracted and copied to the patch folder in the game's installation directory. In Vista this step requires administrative privileges.

<sup>8</sup> This patch was not available from the language patch site, but I located a copy with a search engine.

<sup>9</sup> The nwn2\_pc\_english\* files should only be applied if you have not yet installed the expansion game.



## Patching

When the NWNPatcher tool is run, it opens a dialog window. I selected the Queue button and added each of the RTP files in the exact order listed above. (Note that if you want to allow patch undo, you will have to load the patches one at a time.) Clicking the Patch button launched the patching process, which was successful for me.

After the patch is installed, a set of patch notes can be found in the game's install folder. The patches added many new changes to the toolset. There were also a number of item icons missing, and a couple of the property field comments seemed incorrect. Your mileage may vary, of course, and the tool comes with no warranty. If the patch doesn't work properly you may need to start over and reload the game.

Note that I ran into a problem with the patch:

- `nwn2_pcx1_english_from1221588_to1231763.zip`

that was released in 2009. After installing the patch, the toolset would no longer run even in Windows XP (SP2) compatibility mode. It fails with a problem event name CLR20r3 and a signature 'System.TypeLoadException'.

# Creating a Game

The NWN2 toolset is used to create a role-playing adventure that can be played with the NWN2 game engine. A completed game can range from a short module with a handful of areas up to an extensive campaign spanning multiple modules and many areas. Each module contains a set of interconnected areas that can be traversed by a party of characters, as well as the various area contents, interactive creatures, visual and auditory effects, and the scripts and triggers for processing events. The construction of an adventure requires understanding the toolset's features and functions, many of which are described in this document.

Before you start building a [module](#), it is helpful to have a general idea about the type of adventure you want the players to experience. This game story is formed into a plot that you structure into the module or campaign. The simplest form a game story can take is a linear plot consisting of a set of goals that must be completed in sequence. A more complex plot can employ a branching tree of possible choices by the player, with each branch potentially making significant changes to the choices and outcome of succeeding events.

Until you become familiar with the toolset it is usually recommended that you start simply by creating a few small areas and only allow a limited selection of options for the player. As you gain experience, this initial module could be used as the prelude to a more expansive storyline. For larger games, it may be helpful for you to assemble a basic design document that outlines the story with specific elements and goals. This document may need to be updated as your game is developed, and it can serve as a record so that you can maintain the consistency of your adventure as the game is developed.

## Environment

[Areas](#) form the playing surface that the player will traverse and explore with a group of characters. Each area is constructed by using the toolset editor to create the terrain and decorate it with objects that are rendered as various

structures, furnishings and natural scenery. Some of the objects provide dynamic behavior that can be interacted with by the player, such as [doors](#), [creatures](#), controls, and containers. Other functionality is controlled through marker flags and polygonal outlines that are not directly visible to the player, such as [trigger regions](#) and [waypoint markers](#).

Many of the behaviors of dynamic game objects can be configured by setting their properties. More complex behavior can be managed with the built-in [scripting](#) system. Scripts can be used to manage the animation and reactions of creatures, the effects of [conversations](#), the transition of the party between areas, the special properties of unique equipment, and so forth.

The game experience can be significantly enhanced by visual and auditory effects that will draw the player into the environment and bring the setting to life. The careful selection and placement of [Placeables](#), [Sounds](#), [Lights](#), [Trees](#) and [Placed Effects](#) can all contribute to creating the right mood, as can the area's lighting and weather settings. Likewise, important characters can be customized to present a unique appearance, and their distinctive personality can be represented through conversation interactions. Items can also be customized to present a unique appearance when wielded, thereby contributing to the mythology of the game setting.

## Quests

A game plot can be structured in a modular fashion by defining a set of quests, with each quest having a particular goal and requirements for completion. Quests should be detailed in [journal](#) entries that the player can review during the game. These entries can be updated as each step in a quest is completed, and a reward is often granted upon success. Usually this reward is in the form of experience points, money, or a beneficial item, but it can also include an element that the player needs to advance to the next point in the story, such as a key, a map location, or an important contact.

## Creating a Game

The state of the game's plot is tracked internally using variables that can be set and accessed via [scripts](#). Global variables are stored at the module level and are typically used for plot-related information and game changing events. Local variables are associated with objects, such as a creature or an item, and may be used for tracking specific components of a quest or states of the game environment. In a multi-player module, you would typically want to set local variables on a player's character, because the variables reflect the state of that player in the game.

The completion of a quest requires the player to overcome one or more challenges. Here are some typical obstacles that can be placed in path of a quest's completion:

- [Creatures](#) – The player's PC may encounter creatures that must be overcome by combat, persuasion or some other means. Creatures can be individually placed in an area by selecting them from the Blueprints, or they can be activated by an [Encounter](#) blueprint. In the event of combat, the creature design requires careful balancing by the game designer in order to make certain the player has a reasonable chance of victory, while also suffering a degree of risk. To enhance the player's sense of game depth, some encounters may be configured to allow an alternate approach to success, such as the use of Hide and Move Silently skills to sneak past a powerful guard and steal a key.
- [Conversations](#) – The player may need to hold conversations with one or more creatures and choose the correct paths through the discussion tree. Each response selected by the player may entail a skill check, the possession of a particular item, the completion of a task, a correct decision, or some other factor. Conversations with members of the player's party may also be employed to solve specific problems, or to open up new quests.
- [Skills](#) – The player's path to the quest objective can be blocked by locked doors, traps or hidden features, requiring specific skills to overcome. These obstacles can be set on [Door](#) and [Container](#) objects, or placed at a location via a [Trigger](#) blueprint.
- [Puzzles](#) – These typically require some thought on the part of the player, an element of luck, or both. Solving the puzzle may require the player to orient a set of placeables in a particular order, respond with the correct password in a conversation, answer a riddle, press levers in the right order, see through some form of visual deception, or apply an item that is configured as a key. An area could be configured as a maze using teleports to make navigation more difficult, or the party may be required to pass through a set of doors in a particular order.
- [Exploration](#) – Some quests can simply be about drawing a player into a new location. This area can have obstacles that are not necessarily related to the quest, but provide experience that will ultimately make the player's character more powerful. Some elements of an area can be made inactive until they are needed, or they can be kept hidden until discovered by a successful Search skill check. Transitions between areas can be accomplished through Doors or Triggers that lead to different Doors or Waypoints.

Often there may be more than one way to complete a quest, with the particular path being chosen by the player, or else it can be based upon specific aspects of the player's character type (such as the class or race). Having multiple paths available provides the appearance of depth and makes an adventure seem less linear and predetermined. The most difficult section of a quest is usually saved for the last, such as the final confrontation with the boss of an evil organization or a large scale pitched battle that may require careful planning to win.

As the player advances through the game, they can gain possession of additional gold, gems and [items](#). The acquisition of more powerful items will improve the capabilities of the PCs, allowing them to tackle more challenging quests. However, it is important to balance the power of the items provided with the level of the characters and the degree of risk. At low levels, for example, expendable goods such as potions, scrolls and alchemical mixtures are the more common types of items found.

[Stores](#) are often placed in safe areas of the game

## Creating a Game

environment so that the player can exchange their treasure for useful equipment. Typically a store is opened during an interaction with a creature that is configured as a merchant. The Craft skills can also be used to create useful items, and so suitable crafting tables or similar mechanisms can be made available within the game for this purpose.

### Testing

If you plan to distribute an adventure module to other players, it is essential to perform testing to make certain everything functions as expected. During the construction phases stages you can test the module yourself by using the Run Module item under the File menu. When the game is close to completion, you may also want to ask other players to test it for you in order to catch issues you may have missed.

The game engine automatically assigns you the PC that appears first in the list of pre-generated characters, which is ordered alphabetically by first name. If you don't want to test using the first stock character in the list, you can use the NWN2 game to create and export a customized character that has a suitable name. (For this purpose I created a NWN2 character with a name that begins with "Aa", such as Aaron or Aadela.) Alternatively, the toolset will always select the first '.bic' file in the 'Neverwinter Nights 2/localvault' folder in the documents area of your home folder. To use a pre-existing character, insert a numeral at the start of the corresponding '.bic' file name. This will put the file at the start of the alphabetical sort order.

To test different stages of the plot, you will need to be able to modify variables and make changes to the game. One way to do this is to create a testing script that will make the changes you need when the module is launched. This could be implemented, for example, through the 'On Module Load Script' module property in combination with one or more test parameter set in the module's Variables field. I like to employ a usable placeable that updates the module plot each time it is clicked. Another approach could be to use a conversation as a selection menu. See the chapter on [Writing Scripts](#).

While running the game engine, you can take a screen shot

via Ctrl-Shift-Print Scr. The image is saved as a jpeg file in the 'Neverwinter Nights 2' folder in your documents. The resulting file will be named NWN2\_SS\_*mmddyy\_hhmmss*, where *mmddyy* is the date and *hhmmss* is the time.

### Credits

When you release a module, it is a common practice to list the complete credits for all the work you did as well as help from others and additional content that was added from an online source. One way to do this is to add a [journal](#) entry listing the full credits. A second approach is to create an [item](#) in the starting inventory of the player that contains the credits in the 'Localized Description' property of the item. This can be performed via a [script](#) that is run in the 'On Client Enter Script' property of the first [area](#). A third approach is just to include a separate text file with the module.

## Modules

### Modules

A new, empty module is automatically created when you launch the toolset. If you already have an existing module, you can open it from the File menu. The starting location for a module is set by choosing the 'Set Start Location' in the toolbar, then clicking at a location in an Area. (See the [section on areas](#) for details on how to create interior and exterior areas.)

### Properties

To view the properties of your current module, select Module Properties from the View menu. The properties are displayed in the Properties panel as a set of name-value pairs grouped under heading blocks. The available blocks are Misc, Scripts, Starting Area and Time. The individual fields are described below.

#### Misc

This block includes various global parameters that apply to the module in general.

- **Cached Scripts** – This field is apparently a hold over from the first NWN release and is no longer used.
- **Custom TLK File** – This field is used to provide a customized table for translated text. The property notes recommend this only for advanced users.
- **Description** – This is the text string that will appear when the player first selects the module from the New Module interface of NWN2. It can be used, for example, to specify the recommended character level and to provide a brief hint about the module and its author.
- **Expansion Flags** – The field notes state that this defines the expansions that need to be installed before the module can be played. *I am unclear how this is used.*
- **Hak Paks** – A hakpak is a file that can be used to override the standard content supplied by the vendor. If you use a hakpak, then you will need to distribute it with the modules you develop.
- **Minimum Game Version** – *This field is disabled.*

- **Name** – The name that will appear in the New Module interface of NWN2.
- **NX1 Required** – If true, then the 'Mask of the Betrayer' expansion is required to play this module.
- **NX2 Required** – If true, then the 'Storm of Zehir' expansion is required to play this module.
- **Tag** – The tag used to refer to the module. It is referenced in certain shell commands.
- **XP Scale** – This value is a multiplier for the experience point awards for killing creatures. The default setting of 10 gives rewards that seem in line with the rulesbook values. In a script, this can be changed using the SetModuleXPScale function.

#### Scripts

These event-handling scripts apply across the module, regardless of the area where the players are currently located. Area-specific events should be managed by scripts assigned to each area. Some blueprints also have their own script properties.

Note that some of the fields below were added in patches.

- **On Acquire Item Script** – This script is run whenever an [item](#) is added to the inventory of the controlling PC. Normally it runs the 'x2\_mod\_def\_aqu' script. The GetModuleItemAcquired() routine will return the item that triggered this call. You can use the call GetModuleItemAcquiredFrom() to find the object that lost the item, and GetItemPossessor() to determine the current owner.
- **On Activate Item Script** – This script is run whenever an [item](#) is activated in the inventory of the controlling PC. By default it runs the 'x2\_mod\_def\_act' script. GetItemActivated() returns the item that triggered this script to run. The other GetItemActiv...() calls can be used to obtain more information about the activation.
- **On Client Enter Script** – This runs a script each time a player enters a module. It is empty by default. Use the GetEnteringObject() to determine the player object.
- **On Client Leave Script** – This script is run when a player logs out or leaves the module. By the time this runs, the character has already departed and its data has been removed. The GetExitingObject() call will



## Modules

return the player object that departed.

- On Cutscene Abort Script – The field notes say this is run whenever a player attempts to cancel a cutscene. For example, see 'x2\_abort\_cutscene'.
- On Heartbeat Script – This is run every six seconds while the module is active. In general it is recommended that no heartbeat scripts be set to run continually, or else they will impact performance. A good practice is to minimize the script overhead, usually by exiting early unless the game conditions match the states you want to manage, or if it is running at certain times on the game clock.
- On Module Load Script – This script is run each time the module is first loaded. By default it contains 'x2\_mod\_def\_load'.
- On Module Start Script –
- On PC Loaded Script – This is similar to 'On Client Enter Script', except that you are guaranteed that the PC can be assigned actions.
- On Player Death Script – This is run when a player has died. The default script that is run here is 'nw\_o0\_death'. The `GetLastPlayerDied()` returns the object of the dead PC.
- On Player Dying Script – Each time a PC falls below zero hit points this script is run. By default, the 'nw\_o0\_dying' is run here. `GetLastPlayerDying()` returns the object of the PC that triggered this script to run.
- On Player Equip Item Script – This script is run each time a PC equips an [item](#). By default it runs 'x2\_mod\_def\_equ'. `GetPCItemLastEquipped()` return the item being equipped.
- On Player Level Up Script – This runs once a PC has leveled up. The `GetPCLevellingUp()` call returns the PC object.
- On Player Respawn Script – When a player clicks the respawn button on the death dialog interface, this script is run. `GetLastRespawnButtonPresser()` will return the PC object of the player clicking the button. By default this runs 'nw\_o0\_respawn'.<sup>10</sup>

---

<sup>10</sup> If there is an object or waypoint with the tag

- On Player Rest Script – This is run each time the player decides to rest. By default it is empty, but there is a x2\_mod\_def\_rest script that can be used here.
- On Player Unequip Item Script – Whenever a PC unequips an [item](#), this script is run. The default script is 'x2\_mod\_def\_unequ'. `GetPCItemLastUnequipped()` will return the item that was unequipped.
- On Unacquire Script – This runs when an [item](#) is removed from the inventory of a PC, such as when the player drops an item. The 'x2\_mod\_def\_unaqui' is the default script used in this slot.
- On User Defined Event Script – This script will be triggered by a user defined event (generated by a `EventUserDefined()` call) that is sent to the module by `SignalEvent()`. The module object is returned by the `GetModule()` call.
- Variables – Use this to configure variables that are applicable to the module. If the local integer “X2\_L\_NOTREASURE” is defined and set to true, the automatic treasure generation system (used in the creature spawn in scripts) will be disabled.

### Starting Area

This block contains information related to where the module begins.

- Entry Area – When a starting location has been set, this will list the name of the area where the start was placed.
- Entry Orientation – These four numbers provide the three dimensional orientation of the module start point. Only the last two digits vary when the orientation is varied on the x-y plane.
- Entry Position – This lists an {x, y, z} vector giving the coordinates of the start location in the Entry Area.
- Start Movie – A 'Bink' movie file can be specified here that will run when the module starts.

### Time

This block is used to set chronological information for the

---

“NW\_DEATH\_TEMPLATE”, the PC will appear at that location. See the chapter on Blueprints for information on tags.

## Modules

setting where your module or campaign takes place. These can be left at the defaults unless you want to track the date or modify the planetary parameters.

## Areas

An area forms the playing surface that the party will travel and explore. Each area contains either exterior or interior terrain, and is laid out on a grid surface with fixed dimensions that are set during area initialization. The grid scale differs between interior and exterior areas, with an interior grid square being nine meters across and an exterior grid square spanning ten meters. The surface of an area can be edited with textured terrain or tiles, water-covered regions, and various structures, props, vegetation, creatures and inhabitants.

Exterior areas can have obstructing terrain features and obstacles that channel movement, while interior areas use walls and doors to control flow. Within the game, the player's map will show the resulting area in broad outline, along with various map notes to point out key locations. The game engine constructs this map from a top-down view of the terrain features and any static or environmental objects. Thus it will not show creatures, interactive objects or environmental effects.

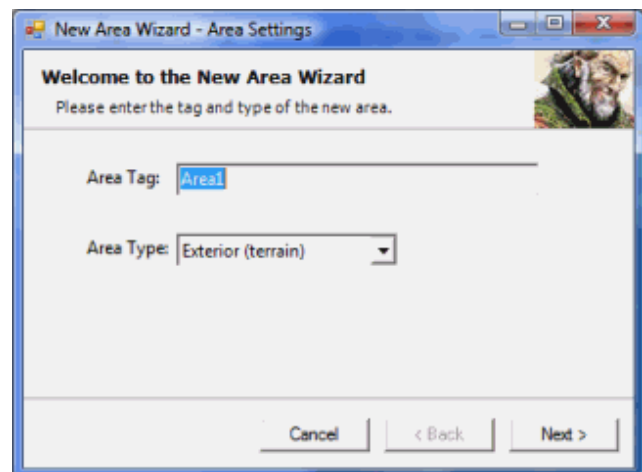
Typically at least one location in an area serves as a transition to a different area. These transition points can be the door to a structure, a cave entrance or just a location. A module requires that a starting point be placed in one of its areas. This location is chosen from the toolbar and placed like as object. (Once this is starting point marker is positioned, the easiest way to select this object is by clicking on the object's arrow head.)

An area can contain various hidden triggers and waypoints that are used to control the spacial aspects of the game flow. Thus a trigger is a region that activate certain events when the players enter, such as revealing new map notes or spawning a group of creatures to do battle. Areas can also have various audio and visual effects to provide a suitable atmosphere.

## Editing

What follows is a summary of how to edit an area. The specific editing techniques for exterior and interior areas differ in several respects, so the unique details for each will be covered in subsequent sections.

To create a new area, you can choose the New item under the File menu, then pick Area from the submenu. Alternatively you can choose the Areas tab on the A/C/S Panel, then right-click with the mouse and pick Add. In either case a dialog window will open that will allow you to choose the basic properties of the new area. You can input a unique area tag, choose whether it is an exterior or interior area, then select the area size as a number of grid rows and columns. The area will then be added to the A/C/S Panel under the Areas tab.



*The New Area Wizard interface*

An area can be opened for editing by selecting it from the A/C/S panel, then right-clicking with the mouse and selecting Open from the pop-up menu.<sup>11</sup> (The same menu can be used to duplicate an area, change an area's name, or remove [delete] an area.) After an interval the area will appear as a tab in the Edit panel, and a rendered view of the

<sup>11</sup> The Toolset will produce an error message if you try to open more than three areas at once.

## Editing

area will appear underneath. If the area includes the start location, the view will open centered on the Start Location object. Otherwise it will be centered on the middle of the map.

Both interior and exterior areas use the same keyboard-mouse controls to manipulate the position of the frustum, or viewing box.

- Drag – holding down the *Ctrl* and the *left mouse button* allows the view to be shifted about horizontally by moving the mouse.
- Rotate – holding down the *Ctrl* and the *right mouse button* allows the view to be rotated and tilted by moving the mouse.
- Zoom – turning the *wheel* on the mouse will zoom in and out of the view.

To close an area, select its tab from the Edit panel, then right click over the tab and select Close [ctrl-C].

If certain features don't seem to be rendering properly in the area, such as the bloom settings under the Day/Night Stages (see below), try exiting the toolset and then running the game normally. In some cases this appears to reset the rendering engine and then the features may start rendering properly again in the toolset.

## Properties

Once an area is open for editing, selecting the area's row in the A/C/S panel will display its properties in the Property panel. These are divided into the Appearance, Environment, Fog, General, Scripts and Sound blocks. The separate blocks are described in detail below.

### Appearance

This block controls how the area will appear as the game clock advances.

- Day/Night Cycle Stages – This can be expanded to display a lengthy set of parameters that can be used to fine tune the area appearance at different points of the day. It is primarily used for the external areas and will be covered in more detail in the [Day/Night Cycle Stages](#) section.
- North Direction – This determines the x-y direction

used for the game's compass north. You can view the direction of north by selecting 'Show North' from the toolbar at the top of the area's edit panel. This will cause a compass to appear at the center of the area map.

- Sky Ring... – these can be used to select the gray backdrop that will appear along the four compass directions of external area horizon. The main menu categories are: large city, small city, forest, island, mountains and none. You should preview these backdrops early in the area building process to determine how you will configure your border terrain features to match.

### Environment

This block has a set of boolean flags that control how the visual appearance will be updated.

- Day/Night Cycle – If this is true, the area will transition between the stages of the Day/Night Cycle Stages as the game clock progresses. When this is set to False, the area will remain at the 'Default' entry of the Day/Night Cycle Stages.
- Directional Light Casts Shadow – If true then structures and other objects cast shadows from the direction of the Sun or Moon.
- Has Directional Light? – This will remove the illumination from sunlight or moonlight.<sup>12</sup> There can still be ambient light and point [light](#) sources such as torches or fires.
- Is Always Night? – If the Day/Night Cycle is set to false, then the area is considered night if this is true; otherwise it is considered daytime.

### Fog

- Use Day/Night Fog Color? – The fog color can be set for different points in the day/night cycle using the expandable Day/Night Cycle Stages fields. If this field is true then the fog color will update as time passes.

### General

- Comments – This field can be used for comments that

---

<sup>12</sup> Setting this property to false can turn off rendering of water surfaces. However, see the [Water Tool](#) section.

## Editing

will not be used in the game.

- Creature Cache – This field is used to select the blueprints of creatures that need to be dynamically spawned, such as by an area's On Client Enter script. In reality it has little effect on creatures spawned in an encounter, as the performance improvement may be measured in milliseconds.
- Display Name – This is the name of the area that will appear on the mini-map during game play. It is also the name used in the A/C/S panel.
- Interior – If true, then the area is considered interior for game rules purposes.
- Load Screen – this entry has a pull-down menu of available screens that will be displayed when the area is loading. The default is Random. Selection of many other screen picks will cause a preview to appear at the bottom of the menu.<sup>13</sup> N2\_Generic\_Default gives a panel with the Neverwinter symbol. The UserDefined is used with the SetAreaTransitionBMP() call to display a custom bitmap.<sup>14</sup>
- Modifier to Listen Checks – This is a circumstance modifier to the Listen skill in this area. It may be appropriate for a noisy environment.
- Modifier to Spot Checks – This is a circumstance modifier to the Spot skill in this area. It may be appropriate for an environment with consistently poor visibility.
- Natural – If true, then the area is considered natural for game rules purposes. Certain spells will only function in natural surroundings.
- No Resting Area – Enabling this will prevent the PCs from resting in the area by default. Selected resting can still be allowed through scripting, which could be used to make certain areas safe for the party. Note that if this property is set to true, then the Module's 'On Player Rest Script' will *not* be run.

- PVP allowed? – This controls whether combat is allowed between players. Full PvP allows combat between friendly and neutral characters; Party Protected prevents members of the same party from attacking each other, and Non PvP only allows damage to be inflicted on enemies.
- Size – These are the grid dimensions as set during the area creation process. Selecting the ellipsis will allow the map to be extended or reduced along any side. If the map is reduced, then the truncated grid areas will be permanently deleted.
- Tag – This name is used to reference the area in scripts.
- Underground – If true then the area will be considered underground for game rules purposes. Certain magic spells can only function above ground.
- Version – This field is disabled.

### OverlandMap

These properties were implemented for the Storm of Zehir expansion and are available after the toolset is patched. Setting the OverlandMap field to true will turn an exterior area into an overland map. There are online tutorials describing the overland map features in detail, so it won't be covered here.

- OverlandCameraDistance – The distance of the overland camera from the map.
- OverlandCameraPitch – The vertical angle of the overland camera.
- OverlandCameraYaw – The rotational angle of the overland camera.
- OverlandMap – Setting this to true will make this area an overland map.

### Scripts

These fields are for various event handling scripts.

- On Client Enter Script – This script is run once the party has completed loading into the area, and is the preferred script to run for processing entering objects (rather than 'On Enter Script'). The main routine must be a function named 'StartingConditional' that returns an integer. You can obtain the entering objects with

---

13 The load screens are stored in the game install directory under UI\default\images\donotatlas and donotatlas\_x1. These files are tabulated in 'loadscreens.2da'.

14 For example, this could be called from the ActionScript [script](#) that is triggered by selecting a Map Point from a [World Map](#).



## Editing

the `GetFirstEnteringObject()` call followed by repeated `GetNextEnteringObject()` calls, until you get an invalid object.

- On Enter Script – This script is run when an object enters into the area. The object can be determined with the `GetEnteringObject()` call. This script should not be used for script operations that depend on the party being present, as there is no guarantee that the full party will be loaded at the time this is run.
- On Exit Script – This script is run when an object exits the area. The exiting object can be determined with the `GetExitingObject()` call.
- On Heartbeat Script – This script is run once every six seconds, regardless of whether any PC is in the area. The presence of a party can be checked at the start of the script to avoid unnecessary overhead.
- On User Defined Event Script – This script will be triggered by a user defined event, generated by a `EventUserDefined()` call, that is sent to the area object by `SignalEvent()`.
- Variables – This field produces a dialog box that can be used to preset variables that are local to the area. The variables can be modified by a script.

### Sound

This section is used to control ambient sounds, music, and the acoustical effects. Note that you can supplement the area's ambient sound set with the placeable [Sounds](#) found under the Blueprints tab of the Selection panel.<sup>15</sup>

- Ambient Sound (daytime) – This field has a menu with a variety of different ambient sound settings that provide a background atmosphere for many situations. The default is to have no ambient sound.
- Ambient Sound (daytime) volume – This setting sets the volume of the daytime ambient sound to a value between zero and 100. If you have the Ambient field set to true under the Sound pick in the toolbar and you change this from the default value, then the selected ambient sound should kick in after a few seconds of load time.

---

<sup>15</sup> Set the sound object's 'Positional?' property to false and set the other properties appropriately.

- Ambient Sound (nighttime) – This is similar to the Ambient Sound (daytime) field, except it is applied at night time.
- Ambient Sound (nighttime) volume – This sets the volume of the night time ambient sound to a value between zero and 100.
- Battle music – This field has a menu of musical tracks that are played when the party is in a combat situation.
- Daytime music – This is the musical track that is played during the day time when the party is not in combat.
- Environmental Audio Effects – This menu pick provides a variety of acoustical settings for different physical environments. This will modify the output of the various sound effects.
- Music delay – This field is a value in milliseconds before a music track begins to repeat. A longer delay may allow faint ambient sounds to be picked up by the player.
- Nighttime music – This is similar to Daytime music, except it is run during the night.

### Weather

These became available after I patched the toolset.

- Chance of Rain – This is a menu that determines how often it will rain in the area. The available options are names instead of percentages.
- Power of Rain – When the rain is active, this determines how intensely the rain falls. However, even at the Stormy rain setting the precipitation may seem like a mild downpour.
- Variation in Rain – The possible values are Large, Medium, Small and None.

### Objects

An area can contain any of the objects chosen from the Blueprints tab of the Selection panel. Most objects have a graphical representation that shows how it will appear within the game setting, and a bounding box that is used for detecting collisions.<sup>16</sup> Note that the number of objects that

---

<sup>16</sup> When editing a blueprint, make sure you haven't got a

## Editing

appear together during the game will determine the graphical performance. Clustering many objects together, especially trees and non-environmental placeables, will place more of a burden on the client's computer.

An object can be added to an area by selecting the row of the appropriate [Blueprint](#) listing. A copy of the object will then appear in the area's Edit panel and will follow the mouse cursor wherever you move it. Pressing the 's' key will toggle stacking mode, causing the object to move to the top of any objects already in the area.<sup>17</sup> To set a copy of the blueprint in place, left click with the mouse. The original object will continue to follow the cursor until you hit the Esc key, thereby allowing you to place additional copies.

### Selecting Placed Objects

In order to select objects, you need to click on the 'Select Objects' tab on the toolbars (or press f2). There are several methods for selecting objects:

- Clicking the left button with the cursor over an object's bounding box will select it and deselect the prior selection.
- Holding down the shift key while clicking objects will add them to the current selections, or remove already selected objects.
- When 'Drag Selection' is enabled in the toolbar, holding down the left mouse button and moving the mouse will create a selection rectangle that picks all objects by their bounding boxes.
- Clicking entries in the Area Contents panel will select the corresponding object in the area.

If you are having trouble selecting an object from amidst a crowded field, you can either try zooming in closer, use the filters in the toolbar, select the object from the appropriate Area Contents list, or some combination of these.

When an object is selected in the view, the cut and copy

---

matching object selected in the area editor. This will cause the local copy to be edited, rather than the blueprint.

<sup>17</sup> Some objects may not stack exactly, and you will need to fine tune the height. An example of this is the 'Table {Wood Rural 02 (X1) TINT}' placeable, where objects auto-stacked on top need to be elevated by another +0.05.

operations in the edit menu items are activated. These allow you to cut or copy the selection, then paste it at a new location (including a different area). You can also use the Delete key to delete the selected object(s), then use undo to restore them back into their previous position. (You may need to move the selected object first before it can be deleted.) The redo item will revert the last undo.

### Translation and Rotation

Selected objects can be moved about by left clicking within their bounding boxes, then dragging the cursor about with the left mouse key held down. For fine tuning an object's position, you can use the arrow keys to move a selected object horizontally in smaller increments; this movement is with respect to the camera position, so you may need to turn the view to get the motion you need.

As they are moved about, objects will adjust their vertical position to remain on the ground or the appropriate distance above it. (Use the Height Lock property to fix the height of an object, or Position Lock to fix the location.) If you want to raise or lower the selected objects by fixed amounts, hold down the shift key and rotate the mouse wheel.

Objects can be rotated about their center with the shift and right mouse key click and drag. If you have a number of objects selected, each will rotate separately about its own center. However, if you right-click and choose Group, then the objects will act as a single unit during translation and rotation. Note that setting the Position Lock property of an object to True will not prevent it from being rotated. This can be a real headache when you are rotating objects in a crowded field and accidentally select the wrong object.

At the top of the area edit panel is a set of tabs for performing object rotations in 45° increments, or resetting an object's orientation.

### Modifying Object Properties

You can modify the properties of an object in the area by selecting it. The object instance will then be displayed in the Properties panel. If multiple objects are selected, only the fields that have identical information will be displayed; the remainder will be blank.

Note that while you are modifying a selected blueprint, a

## Editing

copy of the object will follow the cursor around whenever it appears over the Edit panel. While you are making your edits, you will need to be careful not to click in the Edit panel or press the Esc key.

Once an object has been placed in an area, it takes on an existence of its own. Any changes you make to a blueprint's properties will not be reflected by the objects you have already placed. To synchronize a modified object with a blueprint, you will either have to delete the existing object from the area and place a new copy or else modify the properties of the placed object. In the latter case, you can locate discrepancies between an object and its blueprint because the toolset converts the object's property text to bold font where they differ.

There are two methods for creating a modified copy of a blueprint. When a blueprint is selected in the Selection panel, you can right click on the selection and select Copy Blueprint. Alternatively, you can select an object in the area Edit panel, then right click and choose 'Make Blueprint...'. The blueprint copy will appear at the end of the current node in the blueprint tree.

## Exterior Areas

---

The ground for an external area can be reshaped to resemble virtually any terrain. This flexible surface is formed from a triangular grid that forms a flexible wireframe mesh, allowing the altitude to be adjusted at each grid intersection. The mesh is covered by a texture skin that can be modified by applying various patterns to the surface. To hide the terrain textures and only view the mesh, select 'Wireframe' from the toolbar.

When the 'Grid' option is selected from the toolbar, the area is overlaid by a black grid pattern that forms an array of 20-metre square tiles. These represent the rows and columns of the tile regions that were input in the New Area Wizard. The grid lines are important because toolset applies certain restrictions to each tile, such as the number of textures allowed, and these limits should be a consideration when laying out your map.

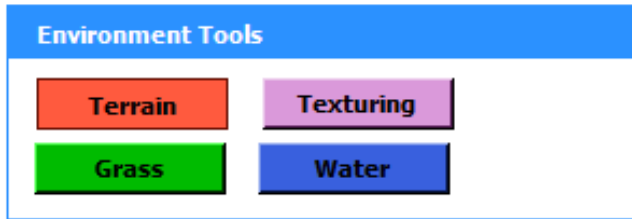
The border of every exterior area has a region, two 20-metre grid squares wide, that is used solely for decoration purposes. In this framing region you create terrain and add objects, but the PCs will not be able to enter here during the game. This border can be used to give the area an illusion of depth by displaying terrain features extending into the distance. However, this border padding also means that the number of row and column grids reserved for the play area is four less than the Size that was input when the area was created. Thus, an area with a width of 16 and height of 12 will only have a player-accessible area of 12 by 8, or 96 tiles.

The area of the surface that is accessible by the players is indicated by a large white rectangle in the Edit panel view. If the 'Occlusion Grid' option is selected in the toolbar, then the sides of this rectangle will be visible through intervening terrain features. Otherwise it can be blocked by the line of sight features, such as ridges.

The surface of an exterior area is edited by a set of tools that are available under the Terrain tab in the Selection panel. These consist of the Terrain, Texturing, Grass and Water tools. Each has its own set of parameters and options that can be set when the corresponding tool is selected. The

## Exterior Areas

tools can only be used on the surface when the Paint Terrain option is selected on the toolbar.



*The four terrain tools*

When the toolset is run in Vista, these options may not be displayed correctly, and the fields will appear to overlapping each other. Per the NWN2 Forums Toolset FAQ, this can be fixed by disabling the application's visual themes as follows:

1. In the Windows Search box, type 'Windows Explorer' and select the tool.
2. Find the NWN2ToolsetLauncher file in the game's install folder.
3. Right click on the launcher and pick Properties.
4. Select the 'Compatibilities' tab.
5. Select the 'Disable visual themes' check box.
6. Click on 'Okay'.

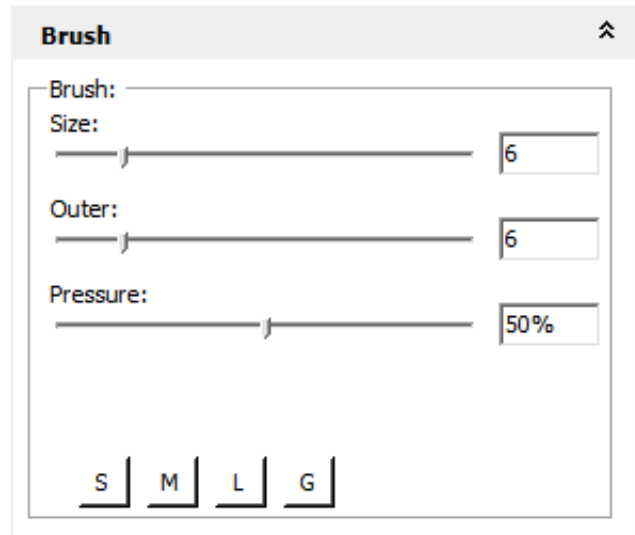
The Terrain editor should display properly once the toolset is relaunched.

### Terrain Tool

When the Terrain tool is first selected with an external area open in the Edit panel, the Paint Terrain option is automatically selected from the toolbar at the top of the toolset. If you use the Ctrl key and click to rotate or move the terrain in the Edit panel, the Select Terrain will become selected. Should you do a Ctrl right-click and drag to rotate the frustum, then the Paint Terrain will be selected again and the brush will reappear. (Thus two consecutive Ctrl right-click rotations will return the brush.)

When the cursor is over the Edit panel with the Paint Terrain mode selected, two concentric orange circles will appear, showing the brush outline. The color of the circles is determined by the tool selected, with orange for 'Terrain', green for 'Grass', violet for 'Texturing' and white for 'Water'.

When the left mouse button is clicked or held with the cursor over the Edit panel, the current Terrain Tool will be applied to the surface under the brush. With 'Terrain' selected, the brush can be used to modify the wireframe mesh.



*The terrain brush settings*

The brush settings are configured in the Brush box on the terrain tool. The Size slider control sets the radius of the inner brush circle to an integer value. (It can be set to zero.) The Outer slider control sets the distance between the inner and outer circles. You can also manually enter digit values in the Size and Outer input boxes, but the fractional part is ignored. When the Flatten tool is selected, an additional Height field is available. You can use a tab or shift-tab to move between these fields; an enter will generate a warning beep.

At the bottom of the brush section are four preset brush sizes: S (small), M (medium), L (large) and G (giant). These use the following dimensions:

Brush Preset	Size	Outer
S(mall)	1	2
M(edium)	6	6
L(arge)	10	10
G(iant)	15	15

The brush pressure setting determines the rate at which the modification is applied to the surface while the left mouse

## Exterior Areas

button is being held down. Pressures of 20% or less are useful for fine changes, while 50% and up produce rapid modifications. Within the inner brush circle, the full pressure is applied to the surface. Between the inner and the outer circles, the pressure decreases linearly with increasing radius until it reaches zero at the outer circle. The tool's effect is applied wherever the brush crosses an intersection on the triangular mesh.

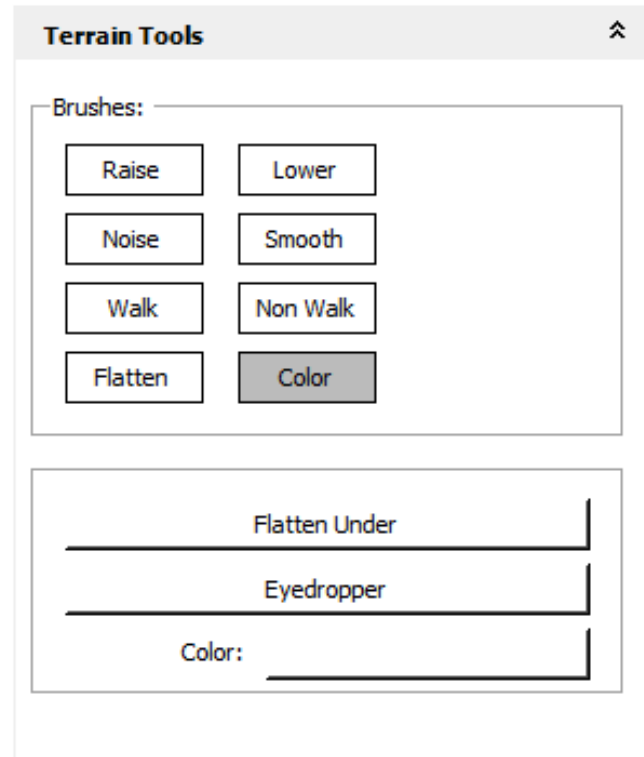
Patch 1.06 introduced several new hot keys for controlling brush size and pressure:

Hot Key	Effect
[	Decrease brush size by 1
shift + [	Decrease outer brush size by 1
]	Increase brush size by 1
shift + ]	Increase outer brush size by 1
-	Decrease pressure by 10%
shift + -	Decrease pressure by 25%
=	Increase pressure by 10%
shift + =	Increase pressure by 25%

There are eight terrain tools, with five being used to adjust the surface mesh.

- Raise – Steadily increase the height of the mesh.
- Lower – Steadily decrease the height of the mesh.
- Noise – Randomly modify the height of the mesh.<sup>18</sup>
- Smooth – Reduce the random variation of the mesh height, effectively flattening the surface.
- Flatten – Set the height of the mesh to the Height.

With this set you can produce virtually any terrain contours you could want to use for your adventure, with the exception of overhangs. However, the terrain height is limited to a range of -100 to 100.



*The tools for modifying the surface contours*

Note that the brush settings are stored independently for each brush type, so switching between the tools can result in a change of brush size.

### Terrain editing

Producing a realistic terrain surface requires practice and patience. Before you begin, it helps to have an idea what you want to achieve. You could, for example, try sketching out the coarse details of the area on grid paper so that you know where to place different features.

A useful method for planning an area layout is to use the Color tool to effectively "block out" the map.<sup>19</sup> To do this, select the Color tool then choose the color tint bar underneath to set the hue. You should select a distinct color with a high visibility, such as red, then paint that hue on the map with a suitable brush size. Different colors can be used to mark where the various roads, structures and features will appear. Don't worry about cleanup at this stage; painting over the surface with a white setting will remove the color

<sup>18</sup> The toolset help warns against using the noise setting across tile boundaries because it can result in breaks between sections. The Smooth tool can be used to repair the breaks.

<sup>19</sup> This is based on a suggestion from a NWN Forum discussion.

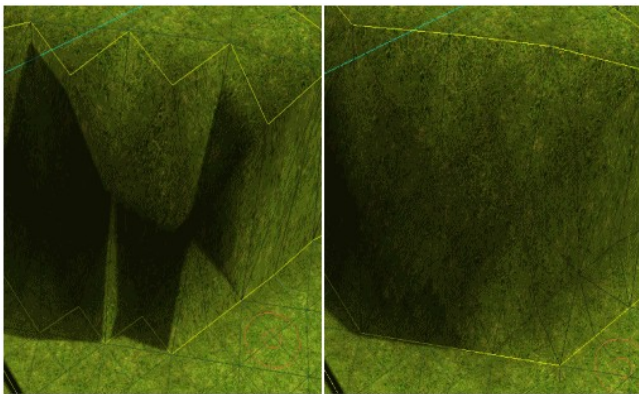


## Exterior Areas

you added.

The shape of the surface you create will depend on the type of terrain you want to build, whether rugged, eroded, or modified by settlers. Areas covered in soil or sand should be softly rounded, while rocky or steeply sloped terrain can be more uneven. Heavily used roads or trails can wear a shallow depression into soft ground. Sea water should be at the lowest elevations of the map, while lakes will form in surface depressions and streams along gently sloping gullies.

A natural rise can be constructed using the Flatten tool to create a series of terraces at the elevations you want to achieve, then fine tuning the surface with the Raise, Lower and Smooth tools. A similar technique can be used to create a road that curves along a rise.<sup>20</sup> Once a rough hill is formed, a more natural appearance can be created by gently adding in the effects of erosion. Water will always follow the steepest slope downward, and will carve out a tree-like pattern of channels that merge into larger stream beds. (See nature photography books for some examples.) In rough terrain this erosion will also sharpen the hill crest, creating ridge lines that radiate outward from the peak.



*An example of fine tuning the surface to eliminate unnatural spikes and dips*

Creating sharply rising ground can result in unnatural-looking sharp points or divots in the surface mesh. These features will become evident when running the module and traversing the ground with a PC, although varying the lighting with the Day/Night toolbar option can also help you

---

<sup>20</sup> Typically, merchant roads will curve around terrain features while military roads will go straight through or across.

find them. These terrain anomalies take patient work with a small brush and low pressure to smooth out.

### Structures

When planning where to place the buildings, note that the typical building structures are going to be smaller than an exterior tile. You can often fit several buildings in a single tile grid. (However, see the section on [Walkmesh](#).)

Most of the buildings and other placeable blueprints don't work well with uneven surfaces. If an area is going to have structures, the surface should be flattened at the locations where the placeables will be inserted. For this purpose you can use the Flatten terrain tool. With the Flatten tool chosen, select the eyedropper and click on the patch of ground where you want to place the building. This will set the Flatten tool's height to the altitude of the terrain at that point. You can then brush over the area to be occupied by the building, and the surface will be set to that height.

An alternative approach is to select a building in the scene and then choose 'Flatten Under' from the terrain tools. This will set the Flatten tool's height to the base of the building. You can then flatten the ground underneath the structure. Note, however, that when a building's height is changed from editing the surface, then the doors may not be set to the proper height. You may need to manually fix this by selecting the doors and moving them back into the openings until they snap into position.

The Flatten tool can be useful for creating farming terraces or a mesa/butte. If I want to create a rising slope for a road, I can use the Flatten tool in a sequence of steps of steadily increasing height to rough out the surface, then polish it off with the Raise/Lower tools. A similar method can be used to create a deep harbor for waterfront scenes. I also find the Flatten tool useful for sinking the ground surface to a level depth under a water-covered area.

### Walkmesh

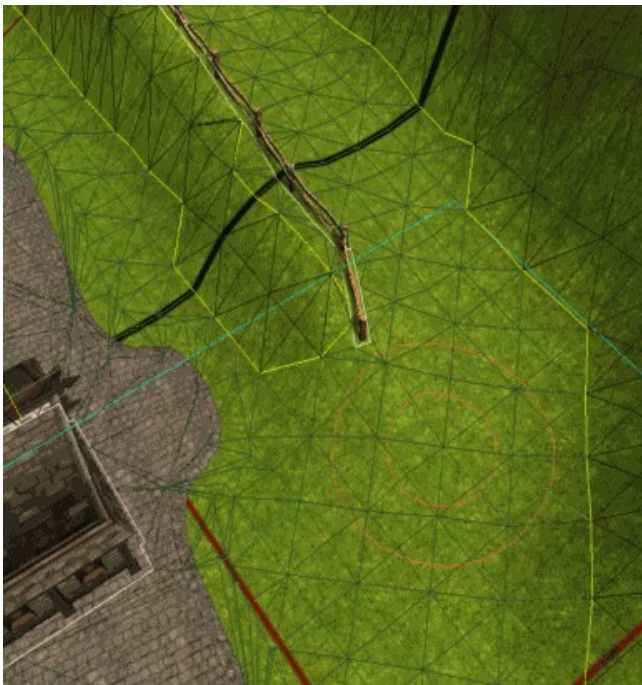
Part of the process for building an area is to determine where the PCs will be able to travel on the mesh grid. This is managed by using the Walk and Non Walk brushes in the terrain tools, followed by a step called baking. When the 'Surface Mesh' is selected from the toolbar, some yellow

## Exterior Areas

lines may appear along the sides of the triangular mesh where the terrain rises sharply. These form the boundaries between walkable terrain and non-walkable terrain. The toolset will automatically make a mesh triangle non-walkable after it reaches a certain inclination (or slope). This occurs at a slope of about 45°.

The baking process is begun by selecting the 'Bake Current Area' item on the File menu. This will cause the toolset to run through a lengthy<sup>21</sup> series of steps to process the area. These include checking for placeable obstructions and walk/non-walk regions, then computing what parts of the map are accessible. The results of a bake can be viewed by selecting 'Baked' from the toolset with the 'Surface Mesh' activated.

*Important: Baking an area is a required step. If you neglect to bake an area, it will appear impassible to the player and it may even crash the game.*

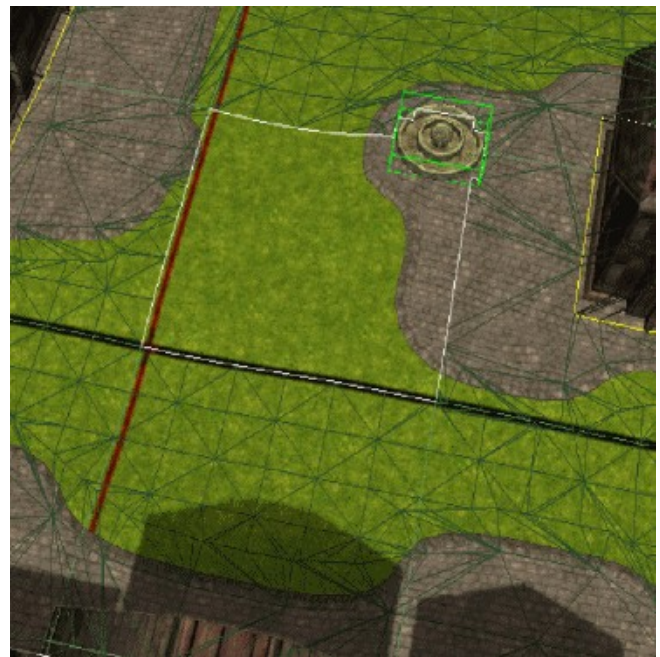


*In this example, the green triangles show the baked mesh, the yellow lines and black triangles define the non-walkable terrain, the thin white lines are the object collision boundaries, and the two orange circles are the brush. The blue lines show the currently active tile.*

Unfortunately, baking an area can result in some

<sup>21</sup> The time required depends on your PC hardware.

anomalies. In sections of the map where placeables are crowded closely together, a bake can completely obstruct movement. This is made visible by a white box around an otherwise walkable location. After baking, it is a good idea to check if any parts of the area have been blocked off in this manner. To fix this problem, you may need to relocate or eliminate some objects then bake again. If small, flat or inaccessible static objects are not expected to obstruct movement, then it can also help to convert these into environmental object. (Environmental object also lower the resources that are needed to run the game, thereby reducing the memory and CPU footprint.) You can also change larger objects to Environmental objects then paint their location with the 'Non Walk' brush. Another option is to use the [Walkmesh Cutter](#), described in the Triggers section of the chapter on Blueprints.



*The white box appeared after baking, showing an open area of the map that has become impassible. In this case, removing the fountain (near top) then baking again fixed the problem.<sup>22</sup>*

After the first cut at baking an area, I like to look it over

<sup>22</sup> Here the problem could also be addressed by locking the curb height settings, converting them to environmental objects, and raising the ground under the curbs until the surface lies just under the placeable's skin.

## Exterior Areas

and check for areas of the mesh that permit player access when those should actually be blocked off. An example of this might be the water's edge or some part of the terrain that you don't want the party to enter, such as swamp or a lava field. This fine tuning can be done with the Walk and Non Walk tools. When Walk is selected, the area under the brush is converted to walkable. Likewise, Non Walk makes the brushed area of the mesh impassible. These adjustments should be performed with Surface Mesh turned on and Baked turned off in the toolbar. Afterward, you will need to bake again.

### Texturing Tool

This terrain tool allows you to paint a pattern onto the surface with the brush. It is the most artistic aspect of area building, and will likely require some practice before you achieve the look you want to accomplish. For this reason it is helpful to experiment on a scratch area. This will allow you to mix and match textures, fine tune the brush settings, and try different approaches.

For the Texturing tool, the pressure setting determines the maximum percentage of the existing texture that will be replaced by a new pattern. Thus, 100% will completely replace the current texture, while a 50% setting will allow half of the prior texture to show through. Gradually varying the setting can be used for blending between one terrain pattern and another.

The texture pattern can be selected from the Terrain Texturing section. This consists of a scrollable list of terrain types that are subdivided into dirt, blight, grass, mud, wood plank, cliff, desert, rocky, sand, snow, and cobble stone patterns. There are also individual patterns for gold, snow cobble, twigs, and plain white, gray and black. Here are some suggestions for terrain use:

- Heavily sloping surfaces will usually be rocky with some scree and twigs along the bottom. The cliff textures are useful for steep slopes, with grass or dirt accumulating at the flatter points. A blend of cliffs can produce a more natural mixture, or the color tool can be used for variation. (See [Color](#) below.)
- Well travelled paths will be worn, with patchy areas

of grass and mud. Likewise, an overgrazed area of ground, or an area of tilled earth, will contain more dirt than grass. In a medieval setting, cobbled roads will be rare outside of settlements and ruins. A road heavily travelled by carts will show parallel dirt or mud tracks.

- An ocean tidal zone can have sand, stained rocks, sea weeds, mud flats and exposed dirt or sand cliffs. Rivers can have sloping banks with mud along the sides and grass along the top. Islands in the river may have piles of rounded stones around the edges.
- Deserts are not always covered in sand. They can have barren, rocky areas and gullies where water may sometimes flood.
- Forest floors are uneven because of roots and fallen trees. In areas where the canopy is thin, the ground can be covered in vegetation. Where the trees are tall and the canopy dense, the ground will be brown from decaying matter. There will be occasional lone rocks and decaying tree stumps.

When a terrain pattern is applied to the surface, it repeats itself in a repeating manner about five times per tile width. This can result in an unnatural appearance when viewed from afar, and you will need to use varying mixtures of blended terrains to conceal this effect. Alternatively, you can place objects or use the color tool to distract from a repeated pattern. Textures are also applied unevenly to slopes, with steep slopes causing the pattern to stretch out in a less than natural manner.

At the bottom of the terrain panel is a Selection Table. This shows all of the terrains that have been applied to a terrain tile, and allows you to select a terrain you have previously applied.

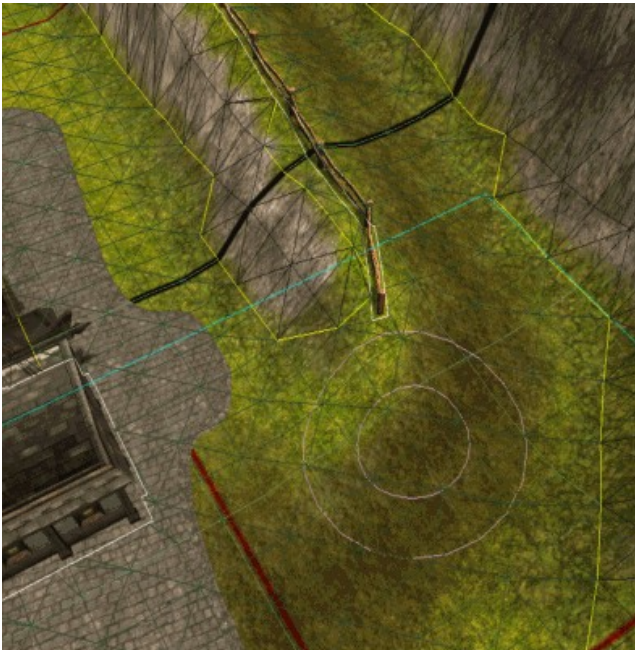
By default an area is covered by the TT\_GG\_GRASS\_19 terrain texture. If you want to change the base texture to something else, first select the terrain type to the new pattern, then click on the 'Fill...' button. You will be prompted to fill then entire area, then, if you click Yes, *all* terrains throughout the area will be replaced. *Warning:* this will eliminate any previous terrains you have set in the region, and reset the number of selected terrain types in



## Exterior Areas

each tile.

Individual textures can be replaced across the map by using the 'Swapper...' tool. First, click on a terrain type in the 'Selection (Advanced)' list. Next, selecting the swapper tool will bring up a dialog interface. On the left side is the texture type you are currently using under the 'Find:' label. A new texture type can be selected from the 'Replace:' list on the right. Clicking Replace will cause texture type in the 'Find:' column to be replaced by the texture type selected from the 'Replace:' list in every tile.



*After replacing the default grass texture a mixture of colors and textures was applied, creating a muddy cliff-side road. Grass and nature props will be added to provide depth.*

As no more than six terrains be applied to a tile, you may suddenly find yourself running out of patterns. For this reason it can be a good idea to plan out what terrain textures you want to use in an area. You will either have to limit yourself to six terrains across an area, or else subdivide the map according to what terrains you will use in various, non-overlapping tile regions.

Regardless of what textures you have applied to the surface, the last texture you use will take precedence over the prior applications. If you want a highlight a particular texture that you have already applied, you can paint over the

surface again with that texture at a suitable pressure setting. This can be done repeatedly, allowing you to layer on previously selected textures until you get the appearance you want to achieve.

### Color

The color setting in the Terrain Tool can be used to modify the tint of the exterior terrain. It can enhance the look of the surface texture, darkening details, enhancing shadows under trees and grass, breaking up repeating textures, creating aged surfaces and forming color gradients. Colors are also useful for tinting underwater surfaces, creating burn scars and adding mysterious stains.

After selecting this tool, you can adjust the color choice by clicking on the Color button. Your color selection will be displayed on the button. The pressure setting will determine the rate at which the color is applied, and low settings can be used to blend colors and apply subtle variations. Use plain white to erase any colors applied to the terrain surface.

While using a small brush size to touch up the colors, you may experience some unevenness in how colors are applied to the surface. This occurs because the colors are applied at the intersections of the triangular mesh, rather than along the lines. Try placing the inner brush so that it is touching one of the intersection points before clicking.

### Water Tool

This tool can be used to create a simulated water surface that displays a constantly changing wave pattern. Normally you would place the water over a depression in the surface, which you can create using the Terrain tool. With the Water tool selected, the brush section of the Selection pane will have a height field that determines the vertical position of the water plane. You can only have one height assigned per major grid square.

Changing the height field then clicking in a grid square will cause all of the water in that square to move to the new height. You can use this capability to create a drop across grid boundaries, where you can place a waterfall using suitable Placed Effects. You can also place lakes, ponds and pools at different altitudes by putting them in separate grid

## Exterior Areas

areas.

Under the brush section is a set of tools in a box labelled Water. This shows the two primary water painting brushes, paint and erase, that can be used to apply or remove the water surface, respectively. The color box determines the hue (H), saturation (S) and brightness (B) of the water. Underneath are several controls that manipulate the basic properties of the water:

- Ripple X/Y – These control how much the water distorts the underlying surface in the X and Y directions. When they are both set to zero, the underwater scene is clearly visible.
- Smoothness – This determines the height of the waves, with a smoothness of 1 being completely flat and free of waves.
- Refraction/Reflection Bias – *Unknown*.
- Refraction/Reflection Power – *Unknown*.

Underneath is a set of three water layers. The appearance of the water is a composite of the three layers, allowing you fine control of the wave movement direction and speed. For a river you will likely want the wave movements to trend downstream, while an ocean or lake will use more random directions. Each layer has two textures, with the first being for a rougher surface and the second producing a more placid wave motion.

Once you have a water surface that you like, you can use the Export button to save it to a file, then Import it later for application in another area.

Note that setting the Area's 'Has Directional Light?' property to false will turn off rendering of water surfaces, except in the grid squares that are within the illumination radius of a point [light](#) source. Setting the Intensity property of a light to zero will remove this illumination, but will still cause the water to render. Thus a single light source with a sufficiently large Range and zero intensity can cause all water surfaces to be rendered.

You can create a water-filled trough by adding the placeable to the scene, then using the Water tool with the appropriate height setting and a small brush. You will need to Paint the water inside the fountain then clean up around the sides with the Erase brush. With rounded forms such as

the fountain, it can be tricky getting it to look decent, so you will need some patience. (Alternatively you can place a circular surface with a single click then scale the placeable to fit.) For less opaque water, try reducing the brightness (B) value of the Color.

## Grass Tool

This tool is used to provide low vegetation to the surface at a cost in extra processing power. Grass can make the landscape appear flourishing and alive, giving it greater depth and interest.

The gently moving grass stalks and leaves are displayed on a series of planar surface placed at various angles to each other. Viewed from above this may seem slightly artificial, but at a low angle it creates a more believable effect. The available fauna types are divided into several short and tall grasses, along with wheat, weeds, reeds, cattails and a few snow growths. The types can be mixed together or placed separately by selecting textures from the list.

The options panel allows you to adjust the size and variation of the selected grass. You can also mix different sizes in the same grid square, so you can combine young and mature growths. For finer control, you can select a smaller brush and use a lower density to apply the grass. Larger blade sizes can be useful for quickly laying down grass in background areas and border grid squares where the player doesn't have access. Large blade sizes combined with tall grass can also be used to create a field of high grass; which is convenient for concealing small opponents.

Each grid square will only allow you to place a fixed number of grass growths, so they need to be used judiciously. Fortunately there is an erase tool that allows you to trim the number of growths so you can better locate the limited numbers. The proper amount to use depends on the surface you are trying to simulate, and how much of the underlying terrain you want to remain visible. Using the minimum amount of grass needed will also keep your module file from growing excessively. If you do need to create a region with thick grass, try placing it at the intersection of grid squares to allow you to use part of the quota from each square.



## Exterior Areas

Grass can be applied sparsely by judicious application. Denser stands of grass will typically grow where there is less traffic, so the areas around fixed objects such as fence posts, structures and boulders are good targets. The player will typically see raised areas before depressions, so placing grass on top of mounds and ridge lines is a good way to break up the sharp edges. To create a realistic patch of grass, I like to scatter short grass in a roughly circular patch, then put a mound of taller grass in the middle. Reeds are good for softening the outline of a pond, and these can be enhanced with lily pads placed just above the water level. To give a rural house a human touch, try scattering a few clumps of flowers to form a small garden.

### Objects

#### Docks and Bridges

Walkable docks and bridges require special care. You don't need to modify the default Properties for these placeables to allow characters to walk across them, but you will need to make sure that the full length of the entry sides are located within a certain distance the ground. Otherwise, the placeable can instead be made an impassible feature during a bake.

For a bridge, the ground at each end must be at about the same height as the vertical position of the placeable. You can make certain of this by using the Flatten feature of the terrain tools with a small brush size. The ground in between the two ends can be any depth as long as the bridge ends are touching the surface. Unfortunately this means that you can not string bridge pieces together unless you put level ground along the line where the bridge pairs join; essentially creating narrow rises at the bridge joins. However, you do have some leeway in the height of the ground. This allows you to set the water height -0.1 below the bridge height, then the intermediate rises to -0.2. Unfortunately this doesn't look very realistic.

An alternative approach is to use the slum docks placeable as a low quality bridge. These can be joined together at their ends and they will bake properly. They need to be lowered about -1.5 so that the ends do not stand above the ground.

Unfortunately dock pieces have a similar problem to the bridge pieces, and even on level ground there is no guarantee that multiple pieces can be joined end to end and still allow the full length to be walkable after a bake. You are limited to either: (1) using the large dock piece, (2) making a dock out of the slum dock pieces, (3) using the [Walkmesh Helper](#) (described in the next section), or (4) repeatedly making fine adjustments followed by another bake. The latter can prove tedious and frustrating.

Other alternatives for waterfront features include inserting a length of raised curb or else using stone wall segments that have been lowered to serve as a breakwater. For the latter to work visually you will need to craft the height of the terrain along the edge so that it drops off sharply. (Roughly a height difference of 10 between the smallest grid nodes gives a decent look.) With careful work this can result in the appearance of a deep-water harbor. Don't forget to add posts for the ships to tie up. If the rural fence posts don't work for this purpose, try scaling a copy of the obelisk to 50% of size and 30% of the height, or scale the stone wall post down to 30% of normal size.



*Wall sections used to produce a river breakwater dock*

#### Walkmesh Helper

Under the Misc Props category of Placeable blueprints are

## Exterior Areas

a pair of objects named Walkmesh Helper. These are invisible surfaces that characters can walk across. Walkmesh Helpers can be used to address baking problems with multi-part docks and bridges, or to create an invisible crossing. Thus, for example, it could be used to create a magical bridge across a canyon, or a walk-across-the-water effect.

The base Walkmesh Helper is a small rectangle at ground level, but it can be increased in size by modifying the Scale field in the Properties panel. There is a Walkmesh Helper for a wooden surface and another for stone, allowing you to choose the appropriate tread noise for the crossing character.

When this object is being used to provide a crossing surface over a bridge or dock, the latter objects should first be converted to an Environmental Object. (Select the placeable, right click then choose 'Convert > Placeables to Environmental Object'.) This will prevent the baking operation from becoming confused. Next, the Walkmesh Helper should be placed just above the bridge or dock, then scaled so it fits the walkable section. (It may help to set the C2 Data on in the Collision toolbar menu so that you can see how things line up.) When you do a bake, the surface will now be walkable. But you may need to appropriately modify the walk mesh around the Walkmesh Helper or you may find the PC walking off the bridge and into the water.

Now rumor had it that the Walkmesh Helper is stackable. However I ran into problems during the bake when I tried to stack two of these items at the same height. The alternative is to use a single large Walkmesh Helper in combination with [Walkmesh Cutters](#) from the Trigger blueprint. The area above the [Walkmesh Cutter](#) will be cut out of the walkable area during the bake. You could use this, for example, when making makeshift, irregular-shaped bridges.

### Day/Night Cycle Stages

The area properties include an expandable field called 'Day/Night Cycle Stages'. This is divided into seven expandable rows covering intervals of a day, plus a default row for use in cases where the Day/Night Cycle is set to false. The DayNightStage[] Array field ellipsis allows you

to read in a Day Night Set File. Each expandable row contains the following fields:

- Bloom – The five bloom settings control the glow surrounding small sources of light, such as a gleam reflecting off metal. This can be used to make a light source appear brighter by spreading it out. However, bloom is processor intensive so it should be used judiciously.



*Bloom Highlight Intensity at 0.54 (left) and 2.16 (right)*

- CloudCover – This is a decimal value between 0 and 1.5 that determines the amount of clouds that appear in the sky. Lower values produce greater cloud cover.
- CloudMovementRateX – This decimal value sets the cloud movement rate in the default east-west direction, with positive values causing movement toward the west. Unless the weather is extreme, you will likely want values of 0.1 or less.
- CloudMovementRateY – This decimal value sets the cloud movement rate in the default north-south direction, with positive values causing motion to the south.
- DesaturateColor – If true then desaturation is active.
- DesaturateFactor – If DesaturateColor is true, then this value controls the amount of leeching of the color while the game is running. It can range from 0.0 to 1.0. To produce a shadow plane effect with almost no color, try turning off the Day/Night Cycle then set the DesaturateFactor value to 0.1 under the Default DayNightStage[] array entry.
- Fog – The game's fog feature can be used to control the range of visibility in an exterior area, reproduce atmospheric haze and, sometimes, to set a particular mood. The expandable fog field determines the thickness, color and distance. The fog's tint is set

## Exterior Areas

using FogColor. The fog will start to appear at a distance FogStart, and reach maximum intensity at FogEnd. To increase the fog density, decrease the FogStart and FogEnd values.<sup>23</sup> Beyond the FarClip distance, the scene will be completely obscured by the fog (thereby reducing the computer's graphics workload). Typically you would want this clipping to occur beyond the FogEnd distance.

- Ground Light – This expandable field determines the illumination of objects using the Diffuse Color setting at a brightness equal to the Intensity. An Intensity of zero eliminates this lighting.
- ShadowIntensity – This is a decimal value between 0 and 1 that determines how dark the shadows appear. Low values can be used to simulate the effect of cloud cover, for example. A value of one gives completely black shadows, while setting the field to zero will eliminate the shadows. Lower values, closer to zero, will make the game run more smoothly.
- SkyDomeModel – This field can be used for an model file (.mdb) that will modify the appearance of the background sky dome. Some examples are the fx\_skydome\_\* models included in the first (X1) game expansion. (You may need to close and re-open the area to see the new effect.)
- SkyHorizon – *Unknown*.
- SkyLight – This expandable field determines the ground surface illumination. The lighting uses the Diffuse Color setting at a brightness equal to the Intensity. An Intensity of zero eliminates this lighting.
- SkyRingColorInterpretation – This decimal value determines how quickly the sky color transitions between the horizon color and the zenith color with increasing altitude angle. Larger values result in a more rapid transition.
- SkyZenith – This sets the color of the sky direction overhead. The color of the sky dome gradually transitions to this color with increasing altitude angle.
- SunCoronaIntensity – The corona is an area of diffuse glow surrounding the Sun. It can be seen when the Sun is low on the horizon. Typically this decimal field has a value of 0.33. Values of 1.3 or larger can make the glow wash out the Sun, giving it the appearance of a giant star. (For the Moon you need a value of 5 or more to achieve this effect.)
- SunMoon – This expandable field sets the color of the Sun/Moon disk to the DiffuseColor, as well as the color of the corona and the reflected glow from the illuminated surfaces and the clouds. The Intensity sets the brightness of the lighting. If SkyLight is set to a low intensity, this will also determine the color of the ground illumination.
- SunMoonDirection – This sets the direction of the Sun or Moon. Clicking on the ellipsis will open up a dialogue that can be used to manipulate the direction. The area scene in the Edit panel will be updated in real time as you maneuver the white arrow about, allowing you to see the effects of the shifting Sun or Moon direction.

---

<sup>23</sup> Setting them equal makes an odd hole-in-the-cloud effect. You can have FogEnd less than FogStart, but it looks unnatural.

## Interior Areas

### Interior Areas

The surface of an interior area is formed by a set of 9-metre square tile patterns that are laid out on a grid. These tiles are selected from a fixed menu of available forms. It is not possible to modify the height of the surface mesh as can be done with an exterior area. However, the tint of many tiles can be modified, and water surfaces can be added.

### Tiles

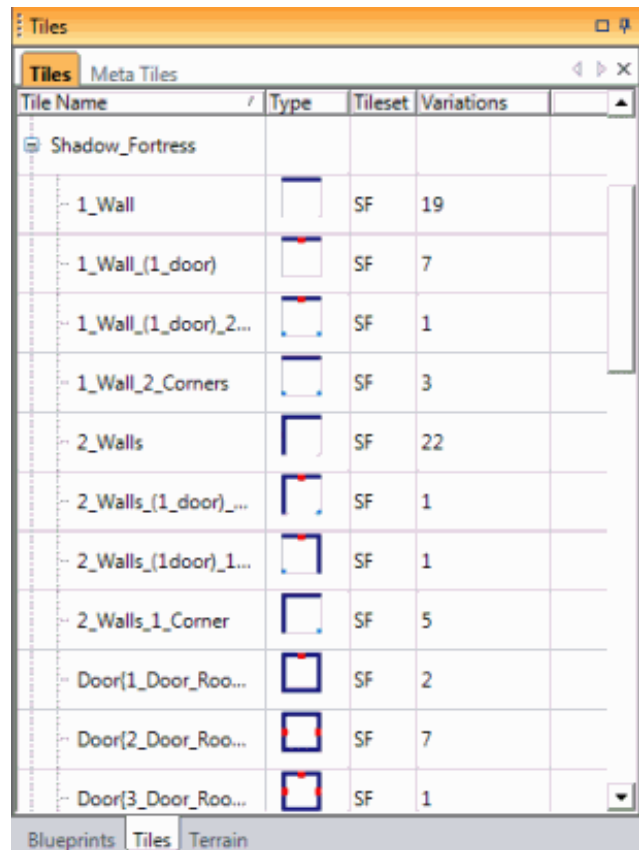
Upon creating a new interior area, the tile grid will be completely filled with black boxes. It is helpful at this point to activate the 'Occlusion Grid' in the toolbar, thereby displaying the tile boundaries. The available tiles can be found under the Tile tab on the Selection panel. This in turn has separate tabs for Tiles and Meta Tiles; the latter are unique forms that can fill multiple tile squares.<sup>24</sup>

The available tiles come in eight styles:

- Standard Interior – Various wooden building interiors with multiple ceiling, wall and floor textures.
- Standard Castle – A stone interior with a single surface style. There are also a few meta tiles.
- Crypt – A set of dingy underground tiles with columns and hanging chains.
- Standard Mine – By default these tiles have dirt sides and wooden supports.
- Caves – These are intended for a natural stone cave system. Some tiles include depressions that can be used for pools, and there are several meta tiles.
- Illefarn – A stone interior with masonry walls that are slightly curved and bevelled.
- Shadow Fortress – A green-tinged stone interior with two floor levels, allowing water-filled areas. You may need to cycle through the tile variations for the elevation you need. Note that the open floor tile only has one variant: the lower level. Thus the upper level is at most two tiles wide. There are several meta tiles

available.

- Sunken Ruins – A dingy stone interior with two levels and gutters along the sides. You may need to cycle through the tile variations for the elevation you need. Note that some of the stair variants allow you to connect the two levels.
- Estate – A marble interior with plain, columned or windowed sides. Various shades of gray tints produce different stone types.



Tile Name	Type	Tileset	Variations
Shadow_Fortress			
1_Wall		SF	19
1_Wall_(1_door)		SF	7
1_Wall_(1_door)_2...		SF	1
1_Wall_2_Corners		SF	3
2_Walls		SF	22
2_Walls_(1_door)_...		SF	1
2_Walls_(1door)_1...		SF	1
2_Walls_1_Corner		SF	5
Door[1_Door_Roo...		SF	2
Door[2_Door_Roo...		SF	7
Door[3_Door_Roo...		SF	1

The tiles listing. Blue lines represent walls, red rectangles are doors, light blue dots are columns or corner pieces, and gray lines are open sides. Note the 22 variations of '2\_Walls'.

To view the available tiles, click on the '-' icon next to the tileset name. This will expand into a list of the standard tile forms. The type column will show the basic profile of a tile, with some combination of light blue dots in the corners for pillars, solid blue sides for walls, red rectangles for doors, and an up or down arrow for stairs. The standard interior and estate tile sets also include roof tiles, which is the black box that initially fills the interior area.

<sup>24</sup> Some Meta Tiles may require a several seconds to load into memory from disk. The toolset may appear to be hung while this is happening.



## Interior Areas

For the purposes of demonstration, try clicking on the '2\_Walls' tile under the standard interior set. This is an L-shaped corner wall about 3/4ths the way through the list. When you move the mouse over the Edit panel, the tile will appear in the grid cell under the cursor. Pressing the left and right arrows (or right clicking the mouse) will rotate the tile, while pressing the up and down arrows will switch between the tile variations. The number of available variations is listed in the same row on the Selection panel.

When you left click in an interior area, the currently selected tile will be placed at that grid cell. You can do this repeatedly, placing the same tile in multiple cells by left clicking and then moving the cursor. (If there is an existing tile in a cell, it will be overwritten by the selected tile.) By rotating the tile with the arrow keys, you should be able to use the corner piece to build a  $2 \times 2$  room.

Within the game, a tile wall will only appear when the player views the tile border from that side. Thus, to build a two-sided wall you need to place a wall on either side. Otherwise, within the game you will have a see-through wall. The doorways are similar in that you must have a matching door opening on each side of the grid face (but you will only need a single door per opening).<sup>25</sup>



*An interior wall before and after adding the corner cap.  
Note the multi-colored hook points along the walls.*

The corner columns can be used to create a cap-piece at an obtuse bend in the wall, which will seamlessly join two right-angled wall segments together. (If you don't complete this step, a gap will show at the bend during the game.) To build an interior column at the joint where four tiles meet, all four tiles must have a column located at the corner where

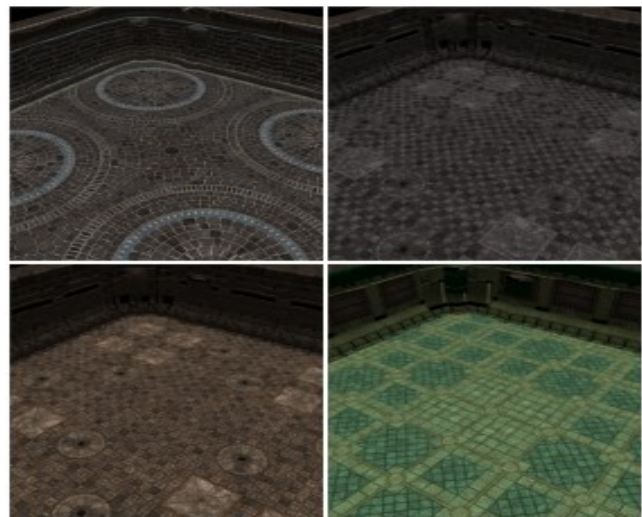
they intersect. Failure to do this will result in a gap along one or more corners of the column. (Thus, for example, filling a two-by-two room with '2\_Walls\_1\_Corner' tiles [properly rotated] will result in a column at the center.)

By default an interior area has the 'Day/Night Cycle' property set to false, so the lighting is determined by the Default entry of the 'Day/Night Cycle Stages'. When the 'Has Directional Light' property is set to true, you can vary the general lighting level by modifying the Intensity value under the SunMoon expandable field in the Default fields. Thus, for example, you could change the Intensity from 1 to 0.5 or less, giving the area a gloomy appearance.

## Tile Properties

In addition to the tile variations, you can modify the appearance of many placed tiles individually by varying the surface textures and tints. These variants can be used to give an interior area a distinctive appearance with a consistent theme, or to vary the look in an area.

You can modify groups of tiles by holding down the shift key and left-clicking the tiles, then modifying the properties in the Property pane. However, this will only work if the selected tiles were already using the same textures and tints.



*Varying the textures and tints can produce  
unique appearances from the same tile set.*

When you select a tile in the Edit panel, the properties of the tile will appear in the Properties panel. These are

<sup>25</sup> The doorways do not come with doors, so you will need to add them in a separate step.

## Interior Areas

subdivided into Appearance and Misc blocks. Each can be contracted or expanded using the small plus/minus box at the left of the header.

### Appearance

- Ceiling Texture – These are the textures that will appear during the game when the character is looking skyward from an interior area. There are several StdRoof textures that are intended for the Standard Interior tiles, a variety of other textures for the other tilesets.
- FloorTexture – These are the surface textures that appear on the interior ground. Using a non-standard floor texture can give your area a unique appearance.
- TintFloor – Some of the floor textures are tintable. You will need to experiment to find out what parts the textures change with each color. Some floor textures will not display properly with certain file sets.
- TintWall – This is similar to the TintFloor, except the tint is applied to the wall textures.
- WallTexture – These are the surface textures that appear on all of the tile's walls. Note that some of the wall textures don't have valid window textures. (Examples: StdWall07 and StdWall08.)

There are issues with certain combinations of textures and tile sets. For example, the Illefarn01 texture does not map properly with the Standard Interior or Standard Castle. Other textures will appear with a multi-colored “2D Missing” message, as with StdFloor03 on a Standard Castle tile. Finally, certain textures will not tint, such as the StdFloor03 in the Standard Interior tiles and the SunkenRuinFloor01 texture with any tile set. The Shadow Fortress tiles will allow you to alter the WallTexture, but modifying the FloorTexture has no effect.

### Misc

Most of the fields in this block can not be edited.

- HookContainerMoved –
- HookedObjects –
- HookPoints – This displays the hook points for interior walls. The list can be viewed but not edited.

- UVScrollFloor – Setting the Scroll field to true in this expandable field will make the floor surface scroll at the rate set by the U and V parameters.
- UVScrollWall – Similar to UVScrollFloor but applicable to the walls. It applies to all sides of the tile.

## Layout

The organization of an interior area will depend on the function it is intended to serve. For a realistic building, there is typically very little internally wasted space, so the rooms and corridors are placed next to each other with no large gaps. (An exception is when a black 'Roof' tile is placed to accommodate a stairwell or multiple fireplaces.) The internal walls of a physical building are used to support the upper structure, and large open volumes may need one or more columns in the middle.

In an underground area, the upper surface is supported by the surrounding rock and dirt, and there will typically be many grid squares filled with black 'Roof' tiles. A natural underground layout will lack the rectangular shape of a building, so it may help to begin with an area that intentionally has excess space along the sides. That is, you should deliberately create an area with more space than you think you will need. This will allow you to make the final layout less boxy, because you will not be as constrained by the edges of the area.

When an interior area is being explored within the game, the player's map will only display the rooms and corridors that can be accessed via an open path from the location of the player. Any closed doors will block this path, so if you want an area to remain hidden until it is explored, then you need to add an opaque door with its Door State set to closed. A barricade across a door opening will not hide the area behind; nor will a gate door.

During an area transition, the player will experience a brief delay as the new area is loaded. For multiple areas within a single building, you can avoid this delay by placing the different floors within a single area. These floors should be separated from each other by at least a single row or column of 'Roof' tiles, and you should make certain that



## Interior Areas

transition tiles have matching orientations. That is, an up stairwell that rises toward the north on the lower floor should link to a down stairwell that descends toward the south on the upper floor. If you don't follow this pattern, then the players may need to rotate their view 180° each time they make the transition.



*This interior area consists of three levels that are linked together by stairwell pairs 'A-A' and 'B-B'. The top floor is at upper left and the bilge is at lower left.*

## Objects

Many of the [placeable](#) blueprints under the Manmade Props classification are suitable for adding decorations, furnishings and containers to interior areas. Balconies are useful if you want to add an elevated section to a room, while the estate tileset placeables are styled to match the Estate tiles. For prisons, try using the Jail placeables. Many of the nature props are suitable for use with the Caves and Standard Mine tiles, particularly those with TINT in the

name. The latter allow their coloration to be modified to match the surrounding tile textures.

As with outdoor areas, you will need to bake an interior area before you use it in a game. In some cases the placement of objects in the area can result in grid blocks becoming impassible. In this case you will have to relocate or remove some objects, then try baking again. My experiences has been that placing objects closer to walls helps alleviate these types of problems, and this also improves path finding during the game. Another method to reduce blockages is to convert flat placeables into environmental objects. Examples of such surface placeables include the floor, floor mat, rug and tarp. Other placeables that can be converted to environmental objects are those stacked on top of other objects, or placeables located behind barriers.



*In this warehouse scene, a complex surface mesh of crates and palettes has been replaced by a single, dark blue Walkmesh Cutter region. This allowed the grid square to bake properly.*

You can use the [Walkmesh Cutter](#), described in the Triggers section of the chapter on Blueprints, to block off a region containing multiple environmental objects. Typically this will be used to mark as impassible a complex area filled with many placeables. The objects to be included in the [Walkmesh Cutter](#) area should be converted into

## Interior Areas

Environmental Objects so that they are not included as part of the bake. For raised surfaces such as a wooden balcony, however, you will not be able to view the [Walkmesh Cutter](#) outline as you are marking around the placeables. Instead, the cut area will only become visible once you perform a bake. Hence, it may take several attempts to get it right.

During a game, the player's map will only display static features. Thus you don't need to worry about concealing [creatures](#) and usable placeables. However, there are times when you may want to keep the true nature of a room concealed until the player reaches that location. One way to do this is to cover a grid square with a static object that is elevated above the player's viewing area. For example, you could position a 'Floor {04}' placeable (under the Manmade Props) at an altitude of, say, +100.0 above an identical placeable on the floor, then group the two placeables together so they don't drift out of alignment and convert them to environmental objects. Any static objects placed on the lower Floor placeable will be concealed on the player's map. Other floor coverings, such as rugs, can be employed to similar purpose.

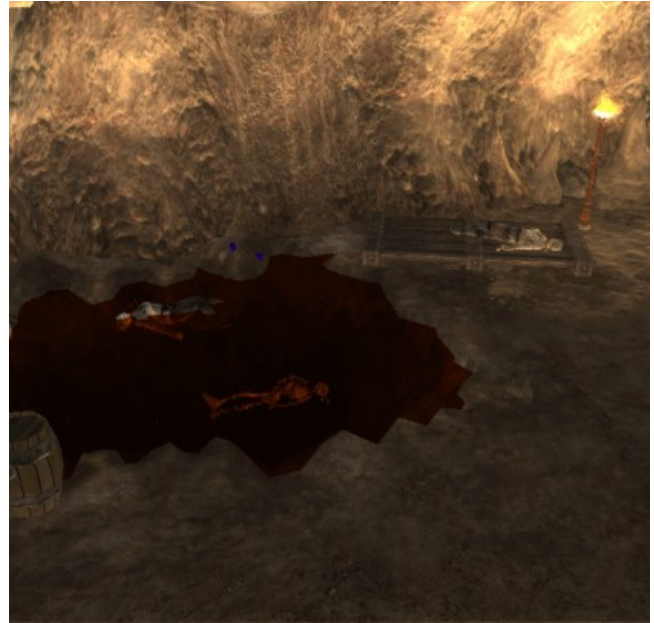
A more drastic approach is to use a 'Tile Block' placeable and scale it to cover a section of the map, such as a secret or concealed room. (You could always add a [map note](#) to mark the location on the map after discovery.) A third option is to use the 'Lid' placeable from the Estate tileset in the manmade props.

### Water

The Water tool under the Terrain tab can be used to add water planes to an interior area. First, make sure that the Water setting is active in the toolbar. Next, select the water tool under the Terrain tab of the Selection window and modify the height to the desired level. For an underwater pool you will probably want to use a high smoothness factor such as 0.98, and low scroll rates. For less opaque water, try reducing the brightness (B) value of the Color.

Certain tiles have build-in depressions that can be used for small ponds, such as variant number 5 of the '2\_Walls' tile in the Caves set. Both the Shadow Fortress and Sunken Ruins tile sets have variants that allow a lower floor level

for use with a pool. The Sunken Ruins tile set includes gullies along the walls where you can put a water layer.



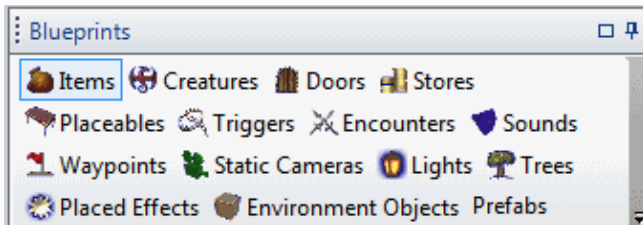
*A grim scene at an underground pool. The water has been given a red tinge to represent blood from the floating corpse.*

Another use for water in an interior area is to simulate a polished floor on tile sets such as Estate. For this purpose, set the Smoothness to 1 and the height to a small fraction above zero (such as 0.02). For the Color, white will provide an untinted, semi-translucent surface. Certain flat placeables may need to be elevated to the same altitude as the water plane so that they appear to rest upon the floor surface. Note also that a water surface will not display shadows and will conceal the outlines of traps.

Within the game, water surfaces require a minimum level of illumination before they will render. If you have the area lightning set to a low intensity, you may find grid squares where the water surface has vanished. You can correct for this by providing point sources of light, or turn on the 'Has Directional Light?' setting.

# Blueprints

The blueprints are a set of objects that can be positioned in an area using the toolset, or generated by a script or a trigger mechanism during the game. To view the blueprints, select the Blueprints tab at the bottom of the Selection panel. The blueprints are organized into categories based on how they are employed within the game. These categories are listed in a tool bar at the top of the Selection panel.



*The available blueprint categories*

The blueprints categories are used as follows:

- [Items](#) – These are moveable objects that can be placed in a creature's inventory, such as various armor, shields, weapons, potions. Items are usually found in the inventory of a creature or container, but they can be obtained from a store or provided as a gift during a conversation.
- [Creatures](#) – These dynamic objects function as creatures within the game. They can be configured to interact using a conversation, and serve as allies or opponents during combat. Creatures can carry and use items.
- [Doors](#) – Various placeables and interior tiles have openings where a door can be inserted. These can be used to control movement between rooms or different areas. Static doors serve as decoration.
- [Stores](#) – These non-visible objects are used to manage the buying and selling of items. They are usually activated through a conversation with a creature, and can include a variable inventory of items. The store settings determined the price at which items are bought or sold.
- [Placeables](#) – This is an extensive list of objects that are used for buildings, walls, bridges, balconies, containers, furniture and decorations. Typically they serve as obstructions that channel movement. Some placeables can be containers that can be locked and trapped.
- [Triggers](#) – This is a non-visible region in an area that is used to activate an event when it is stepped upon by a creature. They can be used for traps or area transitions, for example.
- [Encounters](#) – These blueprints are used to create creatures that will spawn once the PC enters a trigger region on the map. The encounter properties control the numbers and types of creatures that will appear.
- [Sounds](#) – These are various sound effects that can be placed at specific locations and heard within a given radius. They are mainly used to provide atmosphere to the game.
- [Waypoints](#) – These are invisible location markers on an area. They can be used as destinations for area transitions, as map markers, and as places for NPCs to travel.
- [Static Camera](#) – This hidden object creates a marker that can be used to display a particular visual viewpoint during a conversation.
- [Lights](#) – These objects can not be seen directly, but provide point sources of illumination. They can be combined with certain placeables to give the appearance of a lantern or glowing object. You can control the intensity, color and periodic variation of the light.
- [Trees](#) – These are randomly generated foliage that provide the appearance of a tree or bush.
- [Placed Effects](#) – These are dynamic visual effects that can be placed in an area to create points of interest. They can be used to simulate magical or natural phenomenon.
- [Environmental Objects](#) – These are placeables that do not interact with creatures during the game. They take less game memory and CPU processing than other object types.

## Blueprints

- [Prefabs](#) – These blueprints are pre-built groups of placeables that can be inserted into an area for various purposes. They are intended as a time saver while building an area.

When a blueprint category is selected with a left-click, the available selections are arranged in a structured tree list that has a series of expandable nodes. To show the contents of a node, click on the small '+' icon to the left of the node name. Clicking on a '-' icon will contract the node. Note that some nodes can also have several sub-nodes.

A new blueprint can be created by right clicking in a blank part of the Selection panel (or on a tree node) and choosing “Create Blueprint” from the pop-up. If you prefer to make a modified version of an existing blueprint, select the blueprint line, right click and pick “Copy Blueprint”. The copy will appear at the bottom of the same tree node in bold text. You can save a blueprint for use in other modules or campaigns by selecting the blueprint line, right clicking and selecting “Save to File...”, then saving it to the 'override' folder in the NWN2 directory under your local documents.

Once a new blueprint has been created, it's properties can be adjusted to suit your preferences. You can modify these by left-clicking the blueprint and choosing the 'Properties' tab in the Properties panel. Alternatively, right click on the blueprint and select “Properties (new window)”. This will open a separate window with multiple tabs along the top. Many of the properties appear in multiple blueprint categories. However, not all of these properties are used in all blueprints.

The following sections provide details on the various blueprint categories in the order that they are listed in the Selection panel. The environmental objects are not described in detail. These can be used to define placeables that do not need to be converted into environmental objects.

## Items

---

Items are objects that serve as equipment for creatures in the game. Each item has a Base Type that determines how it can be utilized. The blueprint list groups the available items under nodes by their respective Base Types, with the node names giving some indication of their general purpose. At the top level, most items are organized under the Armor, Miscellaneous or Weapons nodes, which are further broken down by sub-type.

When an item is acquired, it is moved to the PC inventory where it is represented by an icon. From there, selected types (such as armor or weapons) can be placed into specific slots in the character's "Paper-Doll" equipment chart based on their Base Type. Hence, an item with a Base Type of 'Armor' can be placed in the player's armor inventory slot.

The unique properties of some items only become active once they are in an equipment slot, while other items only need to be in the character's inventory. Certain item types, such as ammunition or healing kits, become expended after use. These have a stack size, which determines how many identical items can be stacked together in a single inventory slot.

Items can be customized by setting their name, icon, cost, description, unique properties, and the material from which they are formed. Selected item types may display a visual model within the game world when they are placed in the appropriate inventory slot. For example, the armor model can appear about the body of a character when it is moved into the armor slot. The appearance and color of these item models can be fine tuned by selecting the appropriate tab of the blueprint properties.

Magical items can have a description and properties that are not immediately apparent to the player. To determine these properties, a PC must either have ranks in the Lore skill or else use an Identify spell. After an item is identified, an examination of the item will display the revised the description changes and list the properties. These properties may be set to be constantly on, or they may have limitations on their use, such as a certain number of times per day or



## Items

the expenditure of charges. If an item uses charges, it is destroyed once all of the charges have been spent. For information about the standard magic items that are delivered with the toolset, see the '[Named Items](#)' section of the second volume.

Items that are essential to the progress of a game can be flagged as Plot items. This will prevent the item from being sold. Harmful items can be flagged as Cursed, which will prevent the item from being dropped from the inventory (but will not prevent it from being sold).

### Properties

With an item selected, the Properties tab of Properties Panel will display a list of parameter names and values. These are subdivided into Appearance, Armor, Basics, Behavior, Misc and Scripts blocks. Each can be contracted or expanded using the small plus/minus box at the left of the header.

#### Appearance

- Appearance (special effect) – this field can be used to give the item a special effect when it appears in the game world. For example, a sword can be given a flaming edge.
- Container UI Screen – All standard toolset items use the Screen Container Default setting. *Changing this does not appear to impact the game.*
- Icon – This menu contains a long list of icons that will be used when the item is viewed in a PC's inventory. This is a long menu with a lot of graphics, so expect it to scroll slowly, move in jerks and even to appear to hang while the icons load. Many of the icons are grouped together by item type and these have an 'it\_\*' prefix followed by a two- or three-letter code.<sup>26</sup>

---

26 However, there are a number of exceptions: aid potion, ale, ale empty, bowl command water, brazier command fire, broken, brooch shielding, candle, cats grace potion, censer control air, chalice lathander, chime opening, cloak min displace, demon book, elf book, elixir horus re, empty potion oval, generic scroll, gold, healing moss, heal potion, hide leather, ioun stone,

- ac = **clothing**; ah = **heavy armor**; al = **light armor**; am = **medium armor**; as = **shield**.
- be = **belt**; bo = **boot**; br = **bracer**; cft = **crafting item**; ck = **cloak**; cp = ?; ds = sample?; ess = **essence**.
- gem; gl = **glove** or **gauntlet**; he = **headband** or **helm**; key; kit; m = **mold**.
- nk = **necklace**; pa = **parchment**; pot = **potions**; ps = **poison**; qi = **quest item**; ring; s = **spell**; se = **special edition unique weapons**; st = **set**; u = **musical instrument**.
- wa = **axe**; wb = **blade**; wd = **double weapon**; we = blade/other; wm = **magic item**; wo = **ordinance (ammunition)**; wp = **pole weapon**; wr = **ranged**; wt = **thrown**; wu = **blunt**.
- Model Part 1, 2, 3 – These are menus that determine how the item will be displayed in the game when it is being worn or wielded. Not all item types will have multiple parts, and the number of menu picks vary. Some menu items will cause components of the weapon not to be rendered.
- Tint – This expandable field is used to modify the tint of the item parts when it is displayed in the game. The actual effect depends on the base item type and the model parts selected. Many items are not rendered in the game, so they can be left unchanged.

#### Armor

This block will appear in the Properties panel when the item is worn in a chest, helm, boots, gloves, belt or cloak slot. It defines a set of clothing that will be displayed on the creature when loaded into an equipment slot (as long as the creature has its NeverDrawArmor [or NeverDrawHelmet for a helm] property set to false).

- Belt – This is an expandable field that can be used to set the belt's visual type, tints and variation.

---

lantern revealing, lens detection, locked book, malarite totem, mortar pestle, orb elem[ental] summon, plain book, pork jerky, quick silver, quiver, rags, recipe book, scabbard bless[ing], scabbard keen, stone control[ling] earth, torch, white parch[ment], wine, wine empty and yellow parch[ment].

## Items

- Boots – This is an expandable field that can be used to set the boot's visual type, tints and variation.
- Chest – This is an expandable field that can be used to set the main armor visual type, tints, variation, and the various accessories. For belts, boots, cloaks, gauntlets, or helms, this will select the main armor type (cloth, leather, chain shirt, and so forth).
- Cloak – If true, render this cloak when equipped. This should be set to true for items with a Base Type of Cloak.
- Gloves – This is an expandable field that can be used to set the gloves' visual type, tints and variation.
- HasBelt – If true, render the belt when equipped. This should be set to true for items with a Base Type of Belt.
- HasBoots – If true, render these boots when equipped. This should be set to true for items with a Base Type of Boots.
- HasCloak – If true, render this cloak when equipped. This should be set to true for items with a Base Type of Cloak.
- HasGloves – If true, render these gloves when equipped. This should be set to true for items with a Base Type of Gauntlet.
- HasHelm – If true, render this helmet when equipped. This should be set to true for items with a Base Type of Helmet.
- Helm – This is an expandable field that can be used to set the helm's visual type, tints and variation.

### Basics

These fields are mainly used for item identification and description.

- Additional Cost – This is the additional cost of a single item, above that of the base cost. It can be useful for items that have unique properties or special value. Negative values can be also used here to reduce the base cost.
- Base Cost – This locked field shows the item cost as determined by the item type and the selections in the Item Properties field. To get the total cost of the

blueprint, add the Additional Cost *times* the Quantity.

- Base Item – Each item belongs to a base category that determines some of the characteristics during play. For example, an Amulet can be worn in the neck slot, a Potion can be imbibed, and a Gem can not be placed in an inventory slot or be given magical properties. Containers are items that can contain other items. The Base Item type determines the type of item properties that can be applied, the equipment slot where it will be placed, and the sound used when the item is added to the inventory.
- Classification – This determines where the item will appear in the blueprint tree. It is useful for categorizing items that are unique to your module, such as when you want a collection a set of unique items for an area. The string you enter will serve as the base name of the node. Use the pipe character '|' to add sub-nodes. Removing the Classification string will move it to the main items list.
- Comment – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing.
- Identified? – This is the default identified state of an item. When an item has not been identified, it appears blue in the PC's inventory window and the Item properties will not be displayed. Items that have not been identified can not be put in an equipment slot.
- Localized Description – When an item is examined before being identified, this description is displayed in the window.<sup>27</sup>
- Localized Description (when identified) – After an item has been identified, the description in this field takes precedence over the Localized Description entered above.
- Localized Name – This is the name that is displayed

---

<sup>27</sup> To view the default description for a Base Item type, see the 'Description' column of the 'baseitems.2da' file then retrieve the number for the corresponding item type. Use a program such as 'tlkedit' to view the 'dialog.TLK' file in the NWN2 install directory. Find the row entry matching the number. Most of the descriptions are in the range 1667–1736.



## Items

when the item appears in the game. It is listed in stores, during mouse-overs of the item, and in the item examination.

- Resource Name – Is a unique, 32-character name that serves as a blueprint identifier. It is used in the CreateObject function call.
- Tag – This string is often used in script commands for identification and look-up purposes. There can be more than one instance of an object with the same tag.
- Template ResRef – The resource name of the blueprint template that this object inherits from.

### Behavior

- Armor Type (for game rules) – For armor and shields, this determines the game rules effects. The type modifies the base armor class, maximum dexterity bonus, armor check penalty and arcane spell penalty.
- ArmorRulesInfo – Depending on the armor type selected, this locked field will show data from the 'armorrulesstats.2da' file.
- Charges – When an item has Item Properties that use charges, this field specifies the total number of charges that can be used before the charge-based properties can no longer be used.
- Container Preference – For containers, this is supposed to sets the Base Item types that can be stored inside. The available types are: arrow quiver, bolt quiver, gem bag, key ring, no preference (any), scroll case, silver shard bag and sling stone bag. Unfortunately, only the 'No Preference' setting works for this field.<sup>28</sup>
- Cursed – this prevents an item in the inventory from being dropped or transferred, but does not prevent it from being sold. It does not force the item into an appropriate slot.
- Damage Reductions – This is used to specify the Damage Reductions (DR) possessed by the item. The values give an amount of DR, followed by a slash and the weapon types that can ignore the DR.
- Droppable? – this determines whether the item can be taken as loot when the owner is killed.
- Force Into Preferred Container – *No effect.*
- Item Properties – Selecting the ellipsis on this field will display a dialog interface that can be used to set various item properties. The available properties is determined by the Base Item setting. See the [Magic items](#) section below for more details.
- Item Property Activation Preference – This sets the activation policy for item properties. The options are to activate the item when equipped, when in the PC's inventory repository, or either.
- Material – The Damage Reduction (DR) property allows a creature to take less damage from an attack. However, certain materials can penetrate the DR and inflict the full damage. For a weapon, this field can be set to one of these materials, or left as Non-Specific.
- MaximumStackSize – This locked field is the toolset limit for the number of these items that can appear in a single stack.
- Pickpocket – If true, a character can steal this item via the pickpocket skill.
- Plot – If this is true, the item can not be sold at a store. This is used for items that are essential for the advancement of the plot, such as certain keys and various quest items. It's also useful for preventing players from getting rid of cursed items at a store.
- Quantity – This is the quantity of the item that will appear when this blueprint is added to an inventory or a Store.
- Stolen – If true, this is a black market item that reputable shopkeepers will not purchase. Stores can be configured to buy these items (at a markdown) by setting the Store's Black Market property to true.

---

<sup>28</sup> All others will produce an error message when an item is added, saying, "You cannot place items of that type into this container." This occurs, for example, if you try to add gems into a Gem Bag container (even if you set 'Force Into Preferred Container').

### Misc

- UV Scroll – This field can be expanded to show U, V and Scroll slots. If Scroll is true, then the item's surface texture will scroll across the surface at a rate

## Items

determined by the U and V settings.

### Scripts

Items do not have event handler scripts. However, see the ['Item Scripts'](#) section for details on tag-based scripting.

- Variables – This field can be used to define and initialize variables that are used with scripts. Selecting the ellipsis button will open up a dialog window where the variables can be added or modified.

### Base Item

With the Items icon selected in the the Selection panel under the Blueprints tab, you can create a new item by right-clicking and choosing 'Create Blueprint' then 'Module'. This will create a new blueprint with the prefix 'item'. Next, select the item and choose the Base Item type from the Basics section of the properties. This is a pull-down menu of general item categories. The base type determines the equipment slot where the item can be placed. It also determines whether the item will be rendered in the game, and the type of item properties that can be applied.

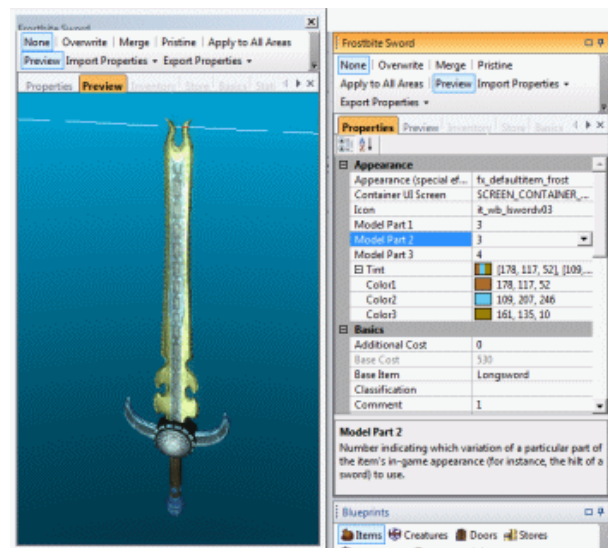
Certain slots only accept a specific Base Item type, such as Armor or Cloaks. Others, such as the various weapons, may accept more than one Base Item type. Finally, there are Base Item types that can not be place in an inventory slot, such as the various miscellaneous objects. If you are not certain what Base Item type to choose, look at some of the available blueprints.

Keys are special purpose items that are used to open specific doors and containers. A key has a base item type of 'Key' and should be given a unique tag. The latter is used in the 'Key Tag' field of a corresponding door or the 'KeyName' property of a container. Typically a door or container that requires a key to open will have the Key Required' property set to true. If opening a door is integral to the story, the key item should have it's 'Plot' property set to true. Various `it_key_*` icons are available for conventional keys, although a key may take another form.

### Appearance

Depending on the base item type chosen, you may be able to modify the appearance of the item model as seen within the game. Items that allow their appearance to be modified are ammunition, armor, clothing, musical instruments, shields, torch and weapons.

A convenient method to edit a weapon's appearance is to bring up it's Properties in a new window (by right-clicking the item) then selecting the window's Preview tab. The Appearance section of the Properties (in the Properties Panel) will then allow you to modify the weapon and view the changes in the window. Some of the items, such as the Long Sword, allow you to customize all three Model parts and to change their tints. By contrast, other items have only a limited selection, and not all of the Model parts menus may produce a visible shape. Several items can be viewed but not modified, such as the arrow.



*Fine tuning the appearance of a magic sword by previewing it in a separate window while adjusting the properties. The Day/Night setting has been modified for better illumination.*

### Armor Set

For armor, first set the Base Item type to the appropriate value for the armor category you want. The 'Armor type (for game rules)' field also needs to be modified to match the specific armor type, or to Cloth for boots, cloak, gauntlets

## Items

or helm.

Items with the Base Item type of Armor, Boots, Cloak, Gauntlet or Helm can now be viewed in the Properties panel under the Armor Set tab. (You may need to scroll to the far right of the tabs using the small triangular arrowhead in the same bar.) To work with the Armor Set, it is easier to select the blueprint, right-click, choose 'Properties (new window)' and select the Armor Set tab.

Note that changing the Base Item type while you are viewing the Armor Set in a separate window can produce uneven results. If you do change the Base Item type, it is recommended that you close and re-open the properties window.

The figure in the window can be selected and moved about as follows:

- Raise/lower – shift-mouse wheel.
- Zoom – control-mouse wheel.
- Rotate the view – shift-right click and drag.
- Drag the view – alt-right click.

If you want to view the figure under different lightning conditions, you can vary the selection under the Day/Night menu on the toolbar. Use the 'Run' and 'Fast' settings to run through a full day's cycle in about 30 seconds.

Depending on the Base Item type, some of the Armor Set fields can now be edited while others will be disabled. For example, with a belt the Main type menu has the single value: Leather, while the Main variation allows options 0-3. You can also modify three tint settings for the belt. (Not all tints will apply to particular item parts.)

### Masterwork Items

The game rules define masterwork items as expertly crafted weapons, armor and shields that provide additional benefits but are not magic items. To create a masterwork weapon blueprint of a long sword, first make a copy of the existing mundane weapon. I change the Classification field to "Masterwork|Weapons", and modify the most of the remaining fields in Basics to distinguish it from the regular longsword. The Identified field should be set to True here, as it is a non-magical item.

For the icon, I look for a matching type that is suitable for a non-magical weapon. For example, if you copy a longsword, the icon is already set appropriately for the item type, so it is just a matter of scrolling down slightly to see if there are any other non-magic icons that can give it a distinctive appearance in the repository. A nice touch is to modify the model parts and tints using the Preview display in a separate window. I like to give the metal parts a slight blueish tinge for a distinct appearance.

A masterwork weapon gains a +1 bonus to attacks, but no damage bonus. Under the Behavior block, select the Item Properties dialog and add a +1 Attack Bonus. This will increase the weapon cost by 200. Masterwork shields and armor gain a slightly different benefit. For an additional 300 gp cost, they reduce the Armor Check penalty by one. This property can be set by selecting the appropriate Masterwork item in the "Armor type (for game rules)" menu under the Behavior section.

Note that under the v3.5 rules, magic armor and shields are also masterwork items. Unfortunately, the stock magic items provided in the toolset are not masterwork rated. If this is an issue for you, then you'll need to copy the items you want and set their armor types to the masterwork equivalents.

### Magic Items

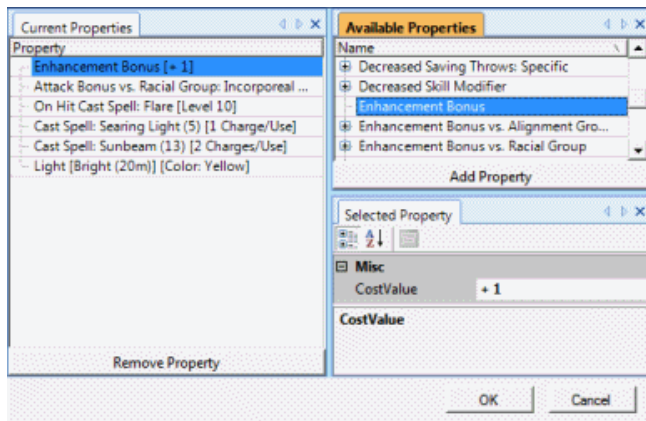
Magical items provide special benefits (or penalties) beyond the mundane properties of a normal item. There are a variety of icons for use with magic items. These can be selected using the pull-down menu in the Icon field. For custom items, I usually avoid the icons used for items that have only a simple enhancement bonus (such as +1 longsword), and I try to select an icon so it will match the preview (where applicable).

Items with magic properties will typically have the "Identified?" field set to True and an extended description in the "Localized Description (when identified)" field. Magic items that are in a PC's inventory when he first joins the party will already be identified. Otherwise, it requires a suitable Lore skill or an *identify* spell to reveal the additional properties.

## Items

The item properties are set using the “Item Properties” interface. To modify the properties, click in the property input box and then select the button with the ellipsis ('...'). The Item Properties dialog has three panels: Current Properties, Available Properties and Selected Property. The list of available properties may vary by the Base Item type, with the Gem and Gold Piece having no properties. Some items have only limited properties available, such as Healer's Kit, Scroll or Trap Kit.<sup>29</sup>

First, scroll through the Available Properties list until you find the property you want to add. In many cases you will need to click on a '+' box to expand a node and view a sub-list. Select a property and then left click the Add Property button.



Configuring the properties of a magic item

Many of the properties allow a range of values. To select the value, click on a property in the Current Properties list and then view the data in the Selected Property panel. Clicking on an input box can present a menu with the valid values. Choose a menu item and the Current Properties entry will be updated accordingly. When the properties are set, click on the OK button. The Base Cost field for the item will be updated to reflect the additional cost of the item properties. More powerful benefits, or those that can be used more frequently, will have a higher cost.

Under the Cast Spell property is a list of spells that can be

cast using the item. Where a parenthetical value exists next to the name, this indicates the caster level of the spell. Thus, *magic missile* (9) is cast at 9<sup>th</sup> level and it provides the maximum allowance of 5 magic missiles. There are also several special purpose spell selections that do not function like normal spells. See the 'Item Properties and Special Abilities' section of the volume II for details.

You may want to control what types of characters can use an item. To do so, modify the Item Properties and add one or more of the Use Limitation properties. Multiple limitations within the same set are generally logical OR'd. Thus an item can have a class use limitation of Cleric and Druid, allowing either class to use it. Note that these limitations can be overcome by a character with a sufficiently high Use Magic Device skill level. (See the game manual for details.)

The following special effects can be useful for wielded weapons, and may be set using the pull-down menu in the 'Appearance (special effect)' field:

Effect	Possible Use
fx_animus	On hit: confusion or deafness
fx_aurora_chain_glow	On hit: sleep
fx_balor_sword	Damage bonus: fire (string)
fx_d_ghost_weapon	On hit: hold or fear
fx_defaultitem_acid	Damage bonus: acid
fx_defaultitem_elect	Damage bonus: electricity
fx_defaultitem_fire	Damage bonus: fire
fx_defaultitem_frost	Damage bonus: cold
fx_defaultitem_holy	Damage bonus: divine
fx_defaultitem_neg	Damage bonus: neg. energy
fx_defaultitem_poison	On hit: poison
fx_e_soul_glow	On hit: daze
fx_familiar_breakup	Damage bonus: pos. energy
fx_githsword	<i>Blue-white streamers</i>
fx_ice_mephit_frost	Damage bonus: cold (minor)
fx_nolaloth	On hit: blindness
fx_pois_dot_linger	On hit: poison or disease
fx_silversword01	<i>Small blue-white tendrils</i>

<sup>29</sup> See the itemprops.2da file for the list of applicable property flags by property category. Select an item column, then press the small triangle for a sort. The cells with a '1' flag the valid properties listed in the far right column.

## Items

### Intelligent Weapon

The toolset can allow you to create a unique weapon that can speak and hold a conversation with the PC. To transform a magic weapon into an intelligent weapon, you will need to do the following:

1. Create a magic weapon with a unique tag and suitable item properties.
2. Add an Item Property "On Hit Cast Spell" with the spell name set to "Intelligent Weapon".<sup>30</sup>
3. Add an Item Property "Cast Spell" with the spell set to "Talk to" and the CostValue set as "Unlimited Uses".
4. Create a non-static Ipoint blueprint with the Template Resref of "x2\_plc\_intwp" and a 'Last Name' matching the intelligent weapon name.<sup>31</sup>
5. Build a suitable [Conversation](#) tree with a name that matches the tag of the weapon.

The default module event scripts, 'x2\_mod\_def\_equ' and 'x2\_mod\_def\_unequ' contain code to check for the Item Property "On Hit Cast Spell: Intelligent Weapon". On success, they call routines from the 'x2\_inc\_intweapon' file that cause the weapon to sometimes speak one-liner strings from its Conversation.

The 'On-Hit Cast Spell' item property can cause random, one-liner strings to be spoken on a successful hit using the weapon. This property is defined by row 135 of the 'irp\_onhitspell.2da' file, which lists a 'SpellIndex' of 768. The 'Intelligent\_Weapon\_OnHit' spell on row 768 of 'spells.2da' has the 'ImpactScript' named 'x2\_s3\_intitemijk'. This runs a command from the 'x2\_inc\_intweapon' include file to sometimes generate a one-liner message.

Adding the item property 'Cast Spell' of type 'Talk to' will allow the PC to hold interactive conversations with the weapon. This spell corresponds to row 536 of the

'iprp\_spells.2da' file, which has a SpellIndex of 767. The spell at row 767 of 'spells.2da' will call the script named 'x2\_s3\_intitemtlk'. This calls an 'x2\_inc\_intweapon' routine to start the conversation. The item property will appear as a "Talk to" menu selection once the weapon is equipped.

When the 'Talk to' menu item is selected, a start conversation routine in 'x2\_inc\_intweapon' will attempt to create a placeable with a resource name of "x2\_plc\_intwp". This placeable blueprint may not exist in your toolset release. But you can construct a suitable item by making a copy of the 'Ipoint' placeable in the MISC PROPS section of the Placeables blueprints. This blueprint should have a Template Resref of "x2\_plc\_intwp", a 'Last Name' matching the name of the weapon, and have a Static property of FALSE.

For the weapon to speak the one-liner messages and hold discussions with the PC, you will need to construct a Conversation for the weapon that has the same name as the item tag. See the [Intelligent Weapon Conversation](#) section for details on how to construct this conversation.

### Cursed Items

To create a cursed item, set the Cursed boolean field to true. This will prevent the item from being dropped or transferred to another character's inventory until a *remove curse* spell is used. Being cursed does not, however, lock the item in an equipment slot or prevent it from being sold. (Note that cursed ammunition or thrown items will also be removed by using them up in combat.) If you want a cursed item to remain in an equipment slot once it is placed there, you can implement a [tag-based script](#).

If you want a cursed item to take effect right away, the simplest approach is to set the Base Item to a type that can not be equipped (such as one of the Miscellaneous\_\* options), then set Item Property Activation Preference to ITEMPROP\_ACTIVE\_REPOSITORY\_ONLY. Next, one or more Item Properties are usually selected that have a negative impact on the owner. Examples include Decreased Ability Score, Decreased Skill Modifier or Weight Increase. (Some cursed items may also have beneficial properties.) To prevent the player from getting rid of the item by selling it at a store, either set the Additional Cost equal to the

---

30 Steps 2 and 3 can instead be applied from a script with the 'IWCreateIntelligentWeapon' call in 'x2\_inc\_intweapon'.

31 Note that the fixed ResRef of this placeable means that there can only be one such intelligent weapon per module. To allow more than one such weapon, you will need to build a modified version of the 'x2\_inc\_intweapon' script, then add duplicates of all the scripts that include this file.



## Items

negative of the Base Cost (so that the Additional Cost would be equal to -9 if the Base Cost were 9), or set the item's Plot flag to true.

### Example

#### Staff of Light

I want to create a staff that has powers of light. To do this, I create a new item and make the following property changes in the order listed:

Property	Value
Base Item	Quarterstaff
Icon	it_wd_qstaff04
Model Part 1	6
Appearance (special effect)	fx_silversword01
Color3	Pure Yellow (#FFF200)
Template ResRef	staff_light
Resource Name	staff_light
Localized Name	Staff of Light
Tag	staff_light
Identified?	False
Charges	99

For the Localized Description ('When Identified') property I enter:

- This staff will bring a cleansing light to the deepest shadows. A magically enchanted length of hickory is topped with a single, clear crystal of quartz that continually glows with a brilliant golden light.

Leaving the 'Localized Description' field entry will cause it to use the default description referenced by {1673} from the 'baseitems.2da' file.

Next I modify the Item Properties as follows:

- Enhancement Bonus [+1]
- Attack Bonus vs. Racial Group: Incorporeal [+2]
- On Hit Cast Spell: Flare [Level 2]
- Cast Spell: Searing Light (5) [2 Charges/Use]
- Cast Spell: Sunbeam (13) [4 Charges/Use]
- Light [Bright (20m)][Color: Yellow]

- Use Limitation: Class: Bard
- Use Limitation: Class: Sorcerer
- Use Limitation: Class: Wizard

Note that I chose to give the item an odd number of charges (99) and each of the charge/use properties an even number of charges (2 or 4). This will prevent the item from being lost due to expending charges. The base cost for this item is 58,852 gp.



## Creatures

### Creatures

The toolset includes a set of pre-built creature blueprints, organized by their racial type and general traits. A creature is a type of object that can have a scripted behavior and move about within an area. The PCs can interact with creatures in the game, whether to hold conversations, pick their pockets, or engage in combat. Each creature has a set of unique statistics determining its capabilities, and these can be modified over the course of a campaign. Creatures can possess an inventory of equipment, such as weapons, armor, gems, potions, and scrolls. Many of the available creatures can be used as is within a module, but you will typically want to customize unique creatures such as NPCs and leader creatures. Player races in particular can require extensive customizing in order to make their appearance less repetitive.

To display the list of available creatures, left-click the Creatures tab at the top of the Blueprints panel. You can either create a creature from scratch or copy an existing creature. To build a new creature blueprint, right-click on the list, select 'Create Blueprint', then 'Module' in the sub-menu. A new, uncategorized entry will be added in bold face text at the bottom of the creature sub-list. To modify an existing creature, select a creature in the list with a left-click, then right-click with the cursor over the creature and choose an option from the Copy Creature menu.

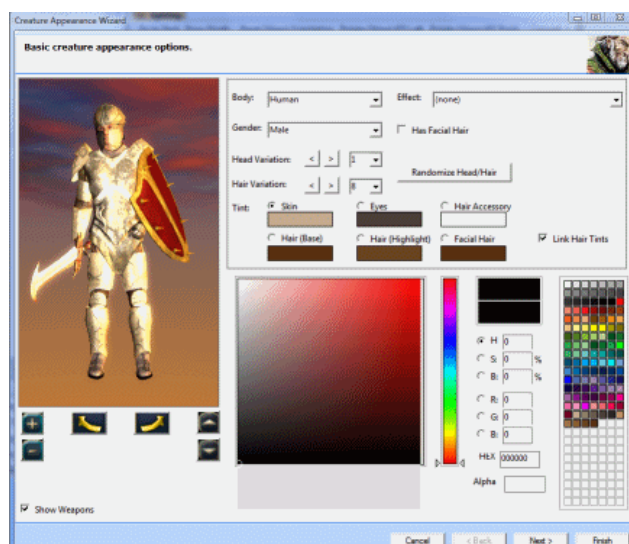
After a new creature blueprint is created, you can select the creature with a left-click then right-click to view the available options for customization. For player races, I typically begin by opening the Appearance Wizard to build a basic look for the creature. When this step is complete, the Properties panel can be opened to modify the creature statistics, feats, skills and special abilities, then add an inventory of equipment.

What isn't immediately apparent is the type of challenge a creature will pose for a PC. For the built-in creatures, you can determine this by looking at the Challenge Rating property. For example, the Mind Flayer creature lists a Challenge Rating of 7, indicating it would be an opponent of moderate difficulty for a party of four fourth-level PCs.

### Appearance Wizard

The Appearance Wizard dialog interface can be used to edit the visual look of your creature within the game. To configure your creature's appearance, select the created blueprint with a left-click then right-click to choose 'Appearance Wizard'.

**Warning:** the use of the rendering pane in the Appearance Wizard can result in unexpected crashes. When creating multiple creatures, it is important to save your module in between each use of the Appearance Wizard. This doesn't prevent the crashes, but at least there was less work lost.



*The first panel of the Appearance Wizard*

### First Panel

Along the left side of this dialog is a rendering panel that shows the current appearance of your creature. Underneath the appearance are controls for moving and rotating the creature image. The '+' and '-' buttons enlarge or shrink the image; the left and right arrows turn it, and the up and down arrows move the image in the corresponding vertical direction.

First, select the Body type from the pull-down menu. You may need to search the list for what you want. Keep in mind that only certain creature types provide the full gamut of appearance options. Thus, you can change virtually every

## Creatures

feature on a player race, but a wolf is very restricted. For maximum variety in appearance, choose a dwarf, elf, gnome, half-elf, halfling, half-orc or human.

For most creatures, setting the Effect to none is appropriate; you can always change it later after the appearance has been set. Next, select the gender. For certain creatures, choosing male or female presents different appearances throughout the editing.

If the Body is set to one of the player races, the head and hair variations provide a mixture of appearances. At this point it is helpful to manipulate the appearance pane to enlarge and center the head and upper body in the view. Rotating the body slightly is useful for viewing the selection at different angles. Try different head and hair combinations until you get what you want, or else just use the 'Random Head/Hair' button. Toggling the 'Has Facial Hair' option may add or remove a beard and/or mustache, depending on the Body type, Gender and Head Variation.

The tint section can be used to provide unique coloration for the creature's head. You can choose colors from the standard palette at the lower right and then fine-tune them by clicking on the saturation/brightness pane at lower center. I find that a pastel red orange or a yellow-orange gives a good starting point for human flesh tones, then fine tune the value in the palette. Selecting a hair base that is darker than the highlight tone results in a natural depth to the hair. For human facial hair, I usually use a tone similar to the hair base, then vary the saturation and brightness slightly. Eye colors are brown, blue or green for humans.

Clicking Next will take you to the second panel.

### Second Panel

The second panel is used to set the creature's default garb. This interface works well for player character races, but has little or no effect for other creatures. I prefer to fine tune the appearance by selecting each of the “Apply To” buttons in turn, then setting the respective appearance. You may want to edit multiple settings at once or do them all together.

To begin fine tuning the garb, I click “Uncheck All” and then select the “Armor” check box. (In the patched version this step is unnecessary.) Next, choose an armor type. What

you do here depends on whether you want the creature to always have a certain appearance, or to show the armor currently being worn in the inventory. If the latter, then you will probably choose “Default (cloth)” for the unarmored look. Some armor types have more options than others, so you will need to experiment to find out what you like.

Next, you can apply tints to the garb. The three tints will apply the selected color to different areas of the armor, although for selected clothing types only some of the tints may function. (The 'Cloth 29' armor, for example, doesn't accept tints.) For the purposes of editing, at times I find I need to select and/or toggle the 'Visible' check box in order to view the selections and tint changes.

There are few options available for showing scantily clad humanoids. The Gloves have a bare flesh option, but the flesh tone here does not always match the tone chosen for the head, so this option is limited. There is no method to display bare feet. One of the cloth armor settings allows for granny's or grandpa's underwear, which looks a little odd for a medieval fantasy setting. The NWN2 Vault has a large number of additional clothing variants, so you may want to explore that web site.

When the armor has been selected and tinted, uncheck the Armor box and choose one of the other “Apply To” check boxes. Rinse and repeat, using the Visible button to toggle the look and untoggling the current selection when applying a new one. Click Next to move to the third panel.

### Third Panel

This panel can be used to fine tune the look of armor by adding various components. If the garb from the second panel will not appear when a character is wearing armor, this panel is not as useful and I usually skip it. For a unique character look, however, this panel is very helpful in fine-tuning the appearance of a humanoid creature.

To begin, you will need to select one of the “Apply To” radio buttons. If your creature will have a symmetrical appearance, “Both” works best. Otherwise you can select different armor for a more unique look. (Note that many of the stock creatures and armor blueprints use asymmetrical armor.)

## Creatures

Armor piece selection begins by choosing one or more of the Affected Attachments(s) check boxes. Some of the boxes may not have variations available, such as the ankle, foot and hip. The shin options will only be available if you did not apply boots on the second panel. Others have only a limited array of options. By contrast, the attachments for parts of the arms, legs and shoulders have a wide variety and you will need to experiment with the variation to find the look you like. You can also tint the armor pieces, much as you did with the basic garb.

When you are done, select Finish. I prefer to do a save immediately afterward in case of a crash. Note that if you go back into the Appearance Wizard after you've exited, you might not be able to modify some of the tints. The work-around is to use the tints menus in the Properties window.

### Properties

With your creature appearance completed, the blueprint can be configured using the Properties Window. After selecting the blueprint entry, right click and select "Properties (new window)". This will open up a separate window with multiple tabs along the top.

### Properties

Selecting the Properties tab will display a list of parameter names and values. These are subdivided into Appearance, Basics, Behavior, Character Sheet and Scripts blocks. When a creature is selected in an area, a Misc block will also appear showing the location information. Each can be contracted or expanded using the small plus/minus box at the left of the header.

### Appearance

This block governs how the character will appear in the game. Many of these are set using the Appearance Wizard: Appearance, Tint and Never Show Armor.

A creature property that I've come to treat with particular caution is 'Appearance'. For example, setting it to 'Half Drow' did not produce a properly rendered character, and even it caused a modification to the appearance of another creature in the scene; whereas 'half-elf' worked fine. In

another instance, I experimented with setting this to an inappropriate value for a human, the 'Dwarf', and thereafter all of the characters were rendered incorrectly in that area: with each showing blond dwarven hair!

Here is a description of the other Appearance fields:

- Body Bag – This is a selectable menu of body bag types. *This setting appears to be ignored.*
- Cast Shadows? – Choose the sources from which the creature should cast shadows.
- Custom Portrait – This menu allows you to select a 128 × 128 pixel Truevision Targa (.tga) file for the creature's in-game portrait. Other image sizes will be scaled to fit.
- NeverDrawArmor – If True, the creature will always show the armor selected using the Appearance Wizard instead of the armor currently being worn. This gets set if you apply the 'Visible' flag in the second panel of the Appearance wizard.
- NeverDrawHelmet – If True, always show the helmet selected with the Appearance Wizard.
- Receive Shadows? – Choose the sources from which the creature should receive shadows.
- Scale – The scale multiplier to apply the the three coordinate dimensions: width, depth and height. This field is useful for creating a set of creatures with a unique appearance, such as people inhabiting a village. Small changes in the range 1–5% usually work best. To make the character tall and lanky, increase the height. For a heavysset character, increase the width and depth. A very portly character may have a width of 1.15 and a depth of 1.35.
- Tail – Apply a tail to the character image. This will only work with certain creature types, especially for those that do not already have a tail.
- UV Scroll – When the Scroll field of this expandable box is set to true, it causes the surface texture to scroll across the creature at the rate determined by the U and V fields. This is used for some of the amorphous creature types such as fire and water elementals.
- Wings – Apply wings to the creature image. The wing selections are intended for specific appearance types,

## Creatures

so they will typically not fit other creatures.

### Basics

This block is mainly used for identification. After a creature is placed in an area, several of these fields are moved to the Misc block.

- **Classification** – This determines where the creature will appear in the blueprint tree. It is useful for categorizing creatures that are unique to your module, such as when you want a collection a set of unique creatures for an area. Use the pipe character '|' to add sub-nodes.
- **Comment** – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing.
- **Conjure Sound Tag** – A tag for a sound blueprint that is intended to be played when the creature is magically conjured as a familiar, companion or summoned monster. It may be unused.
- **Conversation** – This string field can contain the name of a conversation generated with the [Conversation](#) editor. For a non-hostile character, this dialogue is activated by clicking on the creature when the talk cursor is being displayed.
- **Faction ID** – A creature's faction determines its attitude and behavior toward the player. The standard factions are: Hostile, Commoner, Defender and Merchant. Characters with the Commoner faction tend to act cowardly in combat, whereas Hostile creatures readily attack the party. See the Factions menu item under the View menu for a table of inter-factional relations (from 0=hostile to 100=friendly).
- **First Name** – The first name that will appear when the creature is selected in the game. For a PC, this name can be included into a conversation as a label. With generic, non-interactive NPCs, it could instead be used to hold the profession. Thus it could say “Commoner” or “Pirate”, for example. Note that the name can be changed via a script, allowing this field to be updated as the result of a conversation, for example.
- **Last Name** – The last name that will appear when the creature is selected in the game. For a PC, this name

can be included into a conversation as a label. It can be left empty for generic creatures.

- **Localized Description** – This is the text that will be displayed in a panel when the character performs an Examine. It can be used to give significant characters more color.
- **Resource Name** – Is a unique, 32-character name that serves as a blueprint identifier. It is used in the CreateObject function call.
- **Tag** – This string is often used in script commands for identification and look-up purposes. There can be more than one instance of an object with the same tag.
- **Template ResRef** – The resource name of the blueprint template that this object inherits from.

### Behavior

This block determines how the creature object generally functions within the game engine. It does not cover specific creature abilities or event handling.

- **Always Perceivable** – If this is true, then the creature will always be perceivable, regardless of intervening obstacles such as walls or doors.
- **Bump State** – If a creature is bumpable, then it will move aside to allow another creature to pass. An unbumpable creature will hold its position and the game will treat it as an obstruction. The default setting makes non-hostile creatures bumpable.
- **Can Talk to Non-Player Owned Creatures?** – If this is false, then, when an NPC party member initiates a conversation, this creature automatically switches to speak with the party PC. Setting this to true allows the creature to continue speaking with the NPC.<sup>32</sup>
- **Conversation Interruptible** – *Unknown*.
- **Decay Time** – If 'Decays' is set to true, then this is the number of milliseconds until the corpse fades from sight.
- **Decays** – Whether or not the corpse fades away after the decay time.

---

<sup>32</sup> Unfortunately, the 'ga\_open\_store' script in the [conversation](#)'s action section will still use the Appraise skill of the PC for determining costs.

## Creatures

- **Disable AI While Hidden** – Turn off the AI while a creature has the Script Hidden flag set to false.
- **Disarmable** – If true, a creature can be disarmed as a combat maneuver. This defaults to false, but should be true for humanoid and other weapon-wielding creatures (otherwise, why take the Disarm feat?) Unfortunately there is no function call that will allow this setting to be modified in a script.
- **Immortal** – This creature can be damaged, but it can not be reduced below 1 hit points. This could be used for creatures where you want to initiate a conversation prior to their death (followed, say, by a magical teleport or a script that turns the immortal flag to false). You can toggle this value with the SetImmortal call.
- **Lootable Corpse** – If this is true, any inventory flagged as droppable can be looted after the creature is dead. No other items can be looted. Setting the variable X2\_L\_NOTREASURE to 1 on the module will disable the automatic treasure generation system for most creatures.
- **No Permanent Death?** – If true, a dead creature can be brought back to life via a cleric *resurrection* spell. This is useful for potential party members.
- **Perception Range** – This sets the limiting range for this creature to perceive opponents. See the 'ranges.2da' file for more data.
- **Plot** – This makes the creature immune to combat damage or status effects. It generally shouldn't be used for mortal NPCs who could engage in combat, as it makes them invulnerable.
- **Resurrectable** – *Not sure how this differs from 'No Permanent Death?'*.
- **Script Hidden** – If this is true, during game play the creature is treated as though it were not in the game. That is, it can not be viewed or selected, and it is not tracked for collisions. This mode is useful for ambushes or cutscenes. Use the SetScriptHidden call to toggle this state.
- **SelectableWhenDead?** – This causes the creature to be selectable after it is dead, even if it has no loot. This allows the player to perform an Examine of the body.
- **Sound Set** – A long list of sound sets. For NPCs, look in particular at the Female, Human and Male groups. Not every selection gives good results for conversation greetings. You may need to experiment.
- **Spirit Override** – If set, then the creature returns true to the GetIsSpirit call. Elementals and Fey are always considered spirits, regardless of the flag setting.
- **Templates** – These are some of the creature templates found in the D&D monster manual, and they provide modifications to the basic character characters. They are useful for special creature backgrounds, for example, or an outsider version of an animal.
- **Walk Rate** – This is a menu of various walking rates. It defaults to PC\_Movement. The movement rates are listed in the 'creaturespeed.2da' file. The value can be retrieved using the GetMovementRate call.

### Character Sheet

This block includes statistics that would be used if the creature were being used in the tabletop version of D&D. Thus it includes the hit points, armor class, class levels, feats, characteristics and saves. These are more readily configured via the Basics, Statistics and Feats tabs, rather than setting them here.

- **Deity** – This is the name of the deity that appears in the character game panel. It can be retrieved via the GetDeity call. This value can be inserted as a tagged field in conversation strings, and can be checked via the gc\_deity script. The campaign deities are listed in the 'nwn2\_deities.2da' file.<sup>33</sup>
- **Fortitude Save Bonus** – This is the User Modifier bonus to the Fortitude Saving Throw, as set in the Statistics tab.
- **Reflex Save Bonus** – This is the User Modifier bonus to the Reflex Saving Throw, as set in the Statistics tab.
- **Starting Package** – For a party member, this field restricts the Player's choice of class for level up. The SetLevelUpPackage routine can be used to change the

---

<sup>33</sup> Deity selection matters, for example, when using the *recitation* spell.



## Creatures

package, which may be useful for activating a Prestige Class at an appropriate point. A call to the `SetUnrestrictedLevelUp` routine will eliminate the class restriction for the character.

- Will Save Bonus – This is the User Modifier bonus to the Willpower Saving Throw, as set in the Statistics tab.

### Scripts

The Scripts section is used to set event handling scripts and local variables. If you want the creature to behave as they do in the official campaign games, the script fields should be filled in with the appropriate script set. See the [NPC Default Scripts](#) section for details.

- On Blocked Script – this is executed when the movement path of the creature is blocked by a creature, door or object. This script could be used to open or bash a door. The blocking object is returned by `GetBlockingDoor()`.
- On Conversation Script – this is run when the creature is selected for a conversation, or when it hears a shout. The `GetLastSpeaker()` call will return the object that triggered this script.
- On Damaged Script – this script is run each time the creature takes damage. The object causing the damage is returned by `GetLastDamager()`, while the `GetTotalDamageDealt()` call returns the amount of damage.
- On Death Script – when the creature drops below one hit point, this script is run. The `GetLastKiller()` call returns the object that triggered this script. Once a creature is dead, it can not run delayed commands.
- On End Combat Round Script – at the end of this creature's combat round, this script is run.
- On Heartbeat Script – this is always run every six seconds. Custom, time-consuming heartbeat scripts should be used sparingly as repeatedly running them can slow down the game.
- On Inventory Disturbed Script – run this script when the inventory of the creature is changed, such as by a pickpocket attempt or by an item added during a conversation. The `GetLastDisturbed()` call will return

the object that triggered this script to run. The `GetInventoryDisturb...()` calls can be used to find what item was disturbed.

- On Perception Script – each time an object enters the perception range of this creature, this script is run. The object is obtained by a `GetLastPerceived()` call. The `GetLastPerception...()` calls can be used to determine whether the creature will notice the object.
- On Physically Attacked Script – this is run whenever the creature is attacked, whether by melee or a ranged weapon. The `GetLastAttacker()` call returns the attacking object.
- On Rested Script – this script is executed when the creature ends a rest interval.
- On Spawn In Script – run this script when the creature is spawned into an area.
- On Spell Cast At Script – when the creature is the target of a spell, this script is run. The `GetLastSpell...()` calls will return information about the spell.
- On User Defined Event Script – user-defined events will cause this script to run. These are triggered from other scripts by `SignalEvent` calls that are passed an event generated by an `EventUserDefined` function.
- Variables – This field is convenient for setting local variables that are used during conversations with an NPC. Selected variables are used by the standard creature scripts to determine their behavior. See, for example, the various creature variables checked in the default on-spawn handler script, 'nw\_c2\_default9'.

### Inventory

The Inventory tab is used to configure the starting equipment owned by the creature. The upper panel is for items being carried in the non-equipped inventory, while the lower panel summarizes the items in the equipment slots.

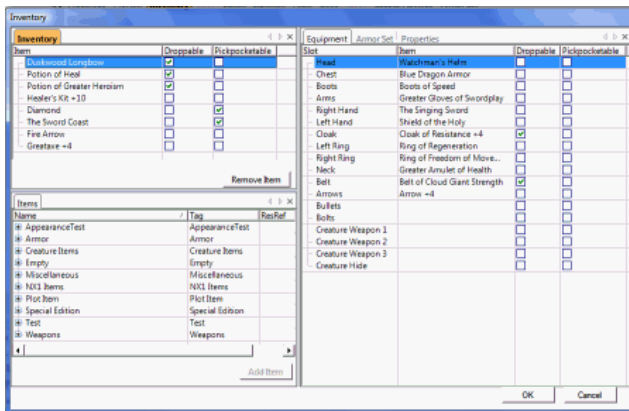
Selecting 'Edit...' will open an editor dialog that will allow you add or remove inventory. The panel in the lower left of the editor dialog contains the same list of items that appears in the Item blueprints. You can select these items and add them to the inventory with the 'Add Item' button. To remove

## Creatures

an item from the inventory, select it and use the 'Remove Item' button.

In the right panel of the dialog is a table of the items that will be placed in creature's equipped items slots. However, in order to use these items, the creature must have the required feats; otherwise the game engine will unequip the unusable items during play. Items can be dragged from the inventory panel in the upper left and dropped in the appropriate slot on the equipped items list. Thus a ring can be placed in the 'Left Ring' or 'Right Ring' fields. To delete an item from a slot, either select it and press the 'Delete' key or else drag it to the Inventory panel and use the Remove Item button.

At the bottom of the right panel is a set of four slots that are used for creature weapons and hide. There are no corresponding slots in the Inventory panel. If a creature with items in these slots is added to a party as an NPC, the natural weapons appear as items in the backpack (and may not even have a valid icon). This can be avoided by only allowing the creature to join a party as a henchman or associate, thereby making the Inventory panel unavailable.



*A sample inventory panel. This creature has five items set as Dropable and two as Pickpocketable.*

When an item is flagged as Dropable in this dialog, it can be looted from the corpse of the creature (assuming you have 'Lootable Corpse' set to true in the creature Properties). Items with the 'Pickpocketable' flag set can be stolen by a successful use of the 'Sleight of Hand' skill. All other items are available to the creature but can not be removed unless the creature joins the player's party as a PC.

### Basics

To the right of the Inventory tab (and the unused Store tab) is a set of tabs for defining the creature character sheet settings. For best results, the Basics, Statistics, Feats and Skills properties tabs should be modified in order from left to right. Thus, Basics should be edited before Feats. If the tabs are used out of order, you may end up with incorrect settings.

First, make certain that the creature has the correct Sub Race, Appearance and Gender. Next, use the Classes section under the Basics tab to define class levels that have been gained by the creature. Each creature race has a favored class, and clicking on the 'Reset From Race Favored' button will set the creature to level 1 in that race's favored class. The favored classes for the PC races are listed in the game manual. A favored class only comes into play when a PC has levels in multiple classes that are not favored classes, in which case the game rules can impose a hefty experience penalty.

To set the initial class, click on the arrow button located on the class name field and choose the appropriate type.<sup>34</sup> If the creature has more than one level in the selected class, increase the value in the field to the right of the class name. The up and down arrows can be used to increment the level. Some classes have a list of default packages, and selecting 'Reset from Package' will set up various feats and skills appropriate for that package and level.<sup>35</sup>

If you want to add an additional class to a creature, click on the Add button then set the class and level for the new addition. The toolset can have as many as four classes, including Prestige Classes. However, script commands such as `GetClassByPosition` only recognize three classes. You should make certain that the requirements for a prestige class are established.<sup>36</sup>

The following table gives a summary of the non-PC racial

<sup>34</sup> The available classes are listed in the 'classes.2da' file, which also lists the hit dice, skill points base, whether it is a player class, and so forth. See the game manual for information on the player classes.

<sup>35</sup> See the 'packages.2da' file.

<sup>36</sup> These can be found in the `cls_pres_*.2da` files.

## Creatures

classes. The base attack bonus increases by +1 per level if the Attack rating is High, by +3 per 4 levels for a Medium Attack rating, and by +1 per 2 levels for a Low attack value. The High saves list the saving throws that are set to +2 at level 1 and increases by +1 at levels that are evenly divisible by 2. All other saving throws are set to the number of class levels divided by 3.

Class	Hit Dice	Attack	High Saves	Skills List	Skill Points
Monster Classes					
Aberration	d8	Medium	Will	Fighter	2
Animal	d8	Medium	Fo/Rf	Cleric	1
Beast	d10	Medium	Fo/Rf	Fighter	1
Construct	d10	Medium	None	Cleric	0
Dragon	d12	High	All	Cleric	6
Elemental	d8	Medium	Fort	Fighter	2
Fey	d6	Low	Rf/Wi	Fighter	2
Giant	d8	Medium	Fort	Fighter	1
Humanoid	d8	Medium	Fort	Cleric	1
Magic Beast	d10	High	Fo/Rf	Fighter	1
Monstrous	d8	High	Rf/Wi	Fighter	2
Ooze	d10	Medium	None	Wizard	2
Outsider	d8	High	All	Rogue	8
Plant	d8	Medium	Fort	Fighter	2
Shapechanger	d8	Medium	All	Fighter	1
Undead	d12	Low	Will	Fighter	2
Vermin	d8	Medium	Fort	Fighter	1
NPC Classes					
Commoner	d4	Low	None	Cleric	1

where Fo=Fortitude, Rf=Reflex and Wi=Willpower.

### Commoners

The Commoner class is an NPC class with significant limitations to skills and feats. In addition, the game AI makes commoners behave in a cowardly manner during combat. They will automatically flee from an attacker for 4 seconds. However, if there is a 10<sup>th</sup> level commoner in the vicinity and he is attacked or killed, then the commoners will form a mob and attack. After taking some losses though, they will disperse. The script notes advise against

making multi-class commoners.

Unfortunately there is no equivalent to the Adept, Aristocrat or Expert classes from the core rulesbooks. Adding these might be possible through modifications to the 2DA files. The Warrior class can be simulated with the Monstrous class.

### Statistics

This tab section configures many of the parameters found in the Character Sheet section of the properties tab. It is divided into sections for Ability Scores, Saving Throws, Hit Points, Armor Class and Attacks. Most of the fields can not be edited, as they are derived from selections on the Basics tab.

The Ability Scores section lists the standard scores used as in the role-playing game. There is no random function or point system for assigning these scores, although the toolset creatures come already configured per the statistics in the game rulesbooks. For humanoid commoners and peasants I like to use a method suggested in the game rulesbooks, which is to assign the sequence 8, 9, 10, 11, 12 and 13 to the various ability scores, based on the profession and background of the creature. Thus a skilled laborer might have Str 13, Dex 10, Con 11, Int 8, Wis 12, Cha 9. For prominent NPCs, I use the sequence 15, 14, 13, 12, 11, and 10 instead. Obviously this is not cut and dried; it is better to use whatever Ability Scores make sense for the creature.

The Saving Throws are computed based on the class, level, ability scores and racial modifiers. However, the 'Use Modified' field can be used to introduce arbitrary modifiers.

By default the total hit points are set to half the maximum per class level, then modified based on the Constitution score. Clicking the 'Reset for MAX' button will recompute the hit points based on the maximum possible value. You can also set the current hit points lower than the maximum, which will cause the creature to appear injured when examined by a mouse-over in the game. If you change the class levels on the Basics tab, the total hit points might not be recomputed. You fix this, will need to select the appropriate Reset button in the Hit Points section on the Statistics tab.

## Creatures

### Feats Tab

This tab section is used to assign various feats to the creature. The top part of the panel has a section for filtering the list of feats. If you change 'Filter By Assignment' to Assigned, then only selected feats will be shown. Any text entered into the 'Filter By Name' field will limit the displayed feats to those that include the text pattern in their names.

Many of the feats are organized into categories as shown in the game manual. The history category lists feats that are used in the core campaign, or during character creation. Racial abilities are the quasi-feats used to represent race-specific abilities, such as Quick to Master for humans. Class abilities are used for the class level special abilities, such as favored enemy for a ranger and the various domains and domain powers for clerics. (Type 'domain' in the Filter By Name to view all the domains.) To display all of the feats, select the 'INVALID\_FEAT\_CATEGORY' setting.

If you clicked "Reset from Race" or "Reset from Package" buttons on the Basics tab, the toolset will select various feats from this list as appropriate for the race or class package. However, if you had previously selected feats from this tab, those may remain selected; resulting in superfluous feats. To fix this you can select 'Remove All', then click 'Reset from Package' on the Basics tab. Alternatively, click the Remove All followed by the 'Add Class Granted Feats', 'Add Race Granted Feats', then select additional feats for the class levels.

### Skills

This table is used for setting the skill ranks in the standard skills used by the NWN2 game engine. Normally the skill points will be assigned for you when you first select 'Reset from Package' on the Basics tab. When I edit this section I like to start by clicking on the Name field. This will place the skills into sort order.

The unfortunate aspect of this tab is that it does not keep tabs of the skill points for you. Instead you will need to work them out manually using the standard game rules manuals. It also does not identify the skills that are class skills or cross-class skills for the character's various class

levels.

You will want to take more care with the starting skills of potential party members, and less so with throw-away characters. If you don't want to work out the skill point details, here are some simple heuristics for quick skill assignment:

- None of the class skills will exceed the total character class levels plus 3.
- Good skills should be based on the character's highest ability scores. A good Charisma score should be matched with a high Cha-based skill, and so forth.
- Bards, outsiders, rangers and rogues will have 6-8 good skills.
- Barbarians, druids, monks, spirit shamans and wizards will have 4-5 good skills.
- All others will have only 2-3 good skills.
- Commoners will have 1 good skill.
- Humans get a bonus high skill.

However, your mileage will vary (considerably). See the game manual appendix for more details.

### Special Abilities

Further to the right on the Properties panel is a tab for special abilities. These are used for various racial specific abilities as identified in the monster descriptions found in the games rules. Many of the picks correspond to spells listed in the game manual. However, there are some picks that have a specific purpose or are associated with an item rather than a creature. For more information on these unique powers, see the 'Item Properties and Special Abilities' section in volume II.

Note that if you add a creature with special abilities to the party as an NPC, those abilities will not appear in the creature's Character Panel. It is only possible to access them by a script that allows the creature to use the ability as a talent. Thus, for example, the Dryad's 'Barkskin' special ability is a talent of type TALENT\_TYPE\_SPELL with an identifier of SPELL\_BARKSKIN.

## Creatures

### Options

#### Custom Behavior

The behavior of creatures is determined by a set of scripts in the Scripts property fields. In order to customize these scripts, you should first become familiar with [writing scripts](#).

You can select a standard script set via the 'Script Set...' menu item under the 'Import Properties' button on the toolbar creature's Properties pane. For example, selecting the `c_StandardScripts` file will install the default scripts used by many of the NPCs and creatures. These scripts are named `nw_c2_default`, followed by a hexadecimal character.<sup>37</sup>

#### Standard 'On Spawn In Script'

The default NPC 'On Spawn In' script is `nw_c2_default9`. This script will check a number of local integer variables to determine the creature's behavior.<sup>38</sup> These variables should be set in the Variables property field for the creature.

The '`x2_inc_switches`' include file defines specific local variables that, when set to 1 on the creature, will trigger certain behaviors and properties when the standard scripts are being used:

- "`X2_L_SPAWN_USE_AMBIENT`" – creature will move about randomly and play animations.<sup>39</sup>
- "`X2_L_SPAWN_USE_AMBIENT_IMMOBILE`" – creature will stay stationary but will perform some animations.
- "`X2_L_SPAWN_USE_SEARCH`" – after spawn-in, the creature will activate detect mode.
- "`X2_L_SPAWN_USE_STEALTH`" – spawn-in using stealth mode.
- "`X2_SPELL_RANDOM`" – make spell use more random, or

<sup>37</sup> For a list of the available script groups, see the '2DA File...' menu item under the View menu, type 'n' and then scroll down to select `nwn2_scriptsets`. The `NPC_Default` scripts set is the same as the scripts in the `c_StandardScripts` file.

<sup>38</sup> For an example, see the Ambient Animations section of the Waypoints blueprints.

<sup>39</sup> See `AnimPlayRandomAnimation()` in the '`x0_i0_anims`' file. The type of random animation performed is determined by the presence of certain waypoints. Thus, a Tavern Waypoint makes the creatures perform tavern-like animations.

even inappropriate.

- "`X1_L_IMMUNE_TO_DISPEL`" – creature will be immune to dispel magic. (Appropriate for statues.)
- "`X2_L_IS_INCORPOREAL`" – creature is able to walk through other creatures.
- "`X2_L_NUMBER_OF_ATTACKS`" – set the number of attacks per round, overriding the default for the BAB.
- "`X2_L_BEH_MAGIC`" – percent chance that the creature will use magic in combat.
- "`X2_L_BEH_OFFENSE`" – percent chance to attack in combat. A value of zero will make them flee.
- "`X2_L_BEH_COMPASSION`" – percent chance that the creature will help others in combat.

If the "`N2_SCRIPT_SPAWN_CREATURE`" global string variable is set to the name of a custom script, this script is run every time a creature is spawned. A local string variable named "SpawnScript" can be set on the creature with the name of a custom spawn-in script. To help build this script, select 'Custom OnSpawn' from the 'Add from Template' menu in the Scripts tab of the A/C/S panel. This template script contains a series of `SetSpawnInCondition` calls that can be uncommented to configure specific behavior. Thus, uncommenting the `X0_COMBAT_FLAG_RANGED` entry will cause the creature to use ranged attacks during combat.

#### User Events

Some of the standard spawn-in condition flags will cause an associated standard script to generate a user event, which can then be handled by a script in the 'On User Defined Event Script' property field. This allows you to use a single custom script to process these events, rather than overriding the individual scripts.

To handle the user events, the 'On User Defined Event Script' property field should be overridden with a custom script. For this, you can use a copy of the standard 'Custom OnUserDefined' template script. The following list shows the property fields that, when set to use the standard scripts, can be flagged to generate user events. The associated user event constant is listed next to the property field name.

- On Conversation – `EVENT_DIALOGUE`
- On Damaged – `EVENT_DAMAGED`
- On Heartbeat – `EVENT_HEATBEAT`
- On Inventory Disturbed – `EVENT_DISTURBED`
- On Perception – `EVENT_PERCEIVE`



## Creatures

- On Physically Attacked – EVENT\_ATTACKED
- On Spell Cast At – EVENT\_SPELL\_CAST\_AT

The 'On Death' script 'nw\_c2\_default7' can be customized by setting a local string variable "DeathScript" to the name of a script. There are no customization features for the standard 'On Blocked', 'On End Combat Round' or 'On Rested' scripts.

### Visual Effects

There are a multitude of visual effects available, but only some of them are useful with the 'Appearance (visual effect)' setting on a creature. Here are some interesting effects to try when you want to create a unique magical, outsider or supernatural creature:

fx_akachi_pers	fx_glowstone_blue_p
fx_animus	fx_haven_song_hit
fx_bugswarm	fx_ice_mephith_frost
fx_defaultitem_*	fx_nolaloth
fx_e_soul_glow	fx_oomany_dematerialize01
fx_e_telepathy02	fx_pois_dot_linger
fx_earth_elemental	fx_reaver_immortal
fx_faithless_golem	fx_shadowfiend_2

For supernatural glowing eyes, try these:

- orange-red: fx\_death\_knight\_eyes, fx\_wraith\_eyes or fx\_lich\_eyes
- blue: fx\_erinyes\_eyes
- yellow: fx\_f\_beetle\_eyes or fx\_hellhound\_eyes
- violet: fx\_helmedhorror\_eyes

If you like flames on a creature:

fx_b_furnace_active	fx_feat_blazing_aura
fx_b_shadow_of_void	fx_fire_mephith_fire
fx_baldor_fire_aura	fx_fireshape
fx_creature_onfire	

Some spell effects can be used for unusual skin coverings:

fx_ironskin_chant_hit	sp_ironbody
fx_moldskin	sp_spiderskin
fx_thayan_golem	sp_stoneskin
sp_barkskin	sp_tortoise_shell
sp_body_of_sun	

See the second volume for a list of effect descriptions.

### Alternate Uses

Many creatures can have their tint changed, giving them a different look in the game. Some creatures can be slightly modified to serve a different purpose.

- Cat: Scale by x2.5 and modify stats for an ocelot.
- Clay Golem: change the skin tint to a fleshy hue, giving it the likeness of a flesh golem.
- Panther: Change the skin/armor tint to white and scale it by x1.1 to produce a near lioness.
- Skeleton: Scale by {2, 2, 1.5} and modify stats for a giant skeleton. Try the various skin coverings for unusual appearances.
- White Wolf: This magical beast can be scaled to x0.5-0.6, stripped of its special ability, and suitably revised to produce a normal sized wolf or dog with a white coat. With the *fx\_animus* special effect, it resembles a devil-dog.

To create one possible interpretation of a sea nymph, I make a copy of the Dryad and used the following settings:

- Appearance (visual effect) – fx\_moldskin.
- First Name – Sea Nymph
- Tag – c\_sea\_nymph
- Inventory – spear, dagger.
- Alignment: Evil/Good: 100; Chaotic/Lawful: 0.
- Challenge: 7
- Ability Scores – Str 10, Dex 17, Con 12, Int 16, Wis 17, Cha 19.
- Natural AC – 0
- Feats – Combat Casting, Dodge, Weapon Finesse
- Skills – Concentration 9, Diplomacy 2, Heal 9, Hide 9, Listen 9, Move Silently 9, Spot 9.
- Special Abilities: Aura of Blinding; Gaze, Stunned; Creeping Cold; Hypothermia; Hold Monster; &c.

Some interesting incorporeal creatures can be created by copying the Mordenkainen Sword under the Special blueprints, removing the sword from the inventory, then setting a continuous and permanent effect in the Appearance (visual effect) property. For example, try these:

- fx\_akachi\_pers
- fx\_blackcloud
- fx\_confusion
- fx\_creature\_onfire

## Creatures

- fx\_defaultitem\_\*
- fx\_ethereal
- fx\_fireshape
- fx\_ghaststench
- fx\_lantern\_archon
- fx\_nshore\_bonfire
- fx\_rockfly\_fade
- fx\_shadowcloak
- fx\_will\_o\_wisp

## Doors

---

Doors are wall or building features that can be used to control access to other locations. When a door is opened, it can act as a transition point between areas, or allow the player to enter another part of an interior area. The available doors are listed under the 'Doors' section of the Blueprints tab, in the Selection panel.

Locks can be used to hinder the player from opening doors, requiring a PC either to bash them down with attacks or else use the Open Lock skill to unlock them. However, an important door can be set to Plot mode, thereby preventing damage from bash attempts. Likewise, a door can be configured to require the use of a key to unlock it, thus disallowing any unlock attempt. Such a key consists of an item that must somehow be located by the player.

A trap can be set to increase the level of risk to the PCs from opening specific door. These traps can be customized by how challenging they are to spot, their level of difficulty, and the amount and type of damage they incur. PCs may spot these traps with the Search skill, then a character with ranks in the Disable Device skill can attempt to disarm the trap or even recover it.

In interior areas, most closed doors obscure the contents of an enclosed space from the player's map. The exception is a door with a barred opening; these are ignored for the purposes of mapping. When a door is opened by the party, the the area beyond the door immediately becomes mapped. When designing an area, this is an important consideration in terms of the information that will be revealed to the player about an area.

## Placement

Many exterior placeables come with doors already fitted in their openings. For example, the 'Wall {City – gate 1}' placeable is equipped with a '{City Gates Double Door}' door. Others have a door opening that initially lacks a door object, but will still allow a door placeable to be selected from the Blueprints list and inserted into the opening. A few buildings have door openings that are not suitable for a door

## Doors

object, such as the 'Windmill (1 – TINT}'. The interior areas have tile walls with empty door openings that can be filled by adding a door placeable.

Because doorway openings vary in size, not every door is going to fit into a particular opening without changing the scale. Within the Blueprints list, the doors are grouped into Exterior, Interior and Walls/Miscellaneous size categories, with the interior doors being somewhat wider than Exterior doors. There are also double doors for wide exterior openings, and secret doors for interiors. Some of the interior doors have barred openings, such as the Portcullis or Gate doors, allowing a view of the area beyond. A few door types have an arched top and are suitable only for certain building openings such as 'Temple {evil}'.

After a door is selected, it can be placed in a doorway opening. The toolset will cause the door to snap into the opening when the cursor is in a certain proximity of the doorway.<sup>40</sup> It can be somewhat tricky to get the door moved into the right location, but moving the cursor to the line along the bottom of the door opening usually works. Once the door is in place, left click to leave a copy in place then press 'Esc'. The placed door can then be selected like any other object in an area.

It is possible to reverse the direction that a door will open by snapping the object to the left or right side of the doorway. This requires selecting a lower corner of the door and placing it to one side or the other of the opening. But it can take patience to get it right. You can also set the door partly ajar by using the Heading field of the properties. The Door State field can be used to fully open the door.

If you are having difficulty selecting a door on a building, try temporarily turning off the 'Placeables' option on the 'Selection' menu in the toolbar. This will prevent the building from being selected by a click, allowing you to choose the door instead.

### Properties

When a door is left clicked in the Selection panel or in an

---

<sup>40</sup> I couldn't find any means to reverse the door orientation so it opens on the opposite edge. Doors only go into the openings one way.

area, it's properties will be displayed as a list of parameter names and values in the Properties panel under the Properties tab. The properties are subdivided into Appearance, Basics, Behavior, Lock, Scripts, Statistics, Transition and Trap blocks. When a selected door is placed in an area, a Misc block will also appear showing the location information. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

#### Appearance

- Appearance – This is a numerical identifier selected from a menu. It determines the door size, shape and texture.
- Casts Shadow – This has no effect because the door does not cast a shadow when opened.
- Door State – By default this is closed, but it can be set to open inward or open outward.
- Load Screen – During a transition, this is overridden by the load screen for the area. (See the Transition block below.)
- ReceivesShadows – This menu item determines the types of light sources that cause shadows to be cast on the door.
- Scale – This can be used to scale a door dimensions.
- Tint – None of the standard doors are tintable, so this can usually be ignored. The exception is for some of the secret doors, which need to be tinted in order to blend into the surrounding surfaces.

#### Basics

After a door is placed in an area, several of these fields are moved to the Misc block.

- Classification – This determines where the door will appear in the blueprint tree. It is useful for categorizing doors that are unique to your module. Use the pipe character '|' to add sub-nodes.
- Comment – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing.
- Conversation – A [conversation](#) can be assigned to a door, but this must be activated from a script. For

## Doors

example, an 'On Open' script could start its conversation with a `ActionStartConversation` function call targeted at the object returned by the routine `GetLastOpenedBy`.

- **Faction** – This is Hostile by default. The setting appears to make no difference as to whether a door can be bashed by the PC or not.
- **Localized Description** – This is the description that appears when the door is examined. If nothing is entered here, a default description will appear.
- **Localized Name** – If the door is not static, this is the name that appears during a mouse-over of the door during a game. Any string within curly brackets '{}' will not be displayed.
- **Resource Name** – Is a unique, 32-character name that serves as a blueprint identifier. It is normally used in the `CreateObject` function call, but that function does not work with doors.
- **Tag** – This string is often used in script commands for identification and look-up purposes. There can be more than one instance of an object with the same tag. A unique tag is recommended when the door is used as a transition point. (See the Transition block below.)
- **Template Resref** – The resource name of the blueprint template that this object inherits from.

### Behavior

- **Can Talk to Non-Player-Owned Creatures?** – *Unknown*.
- **Interruptible** – *Unknown*.
- **Plot** – If true then the door can not be damaged, destroyed, or afflicted with status effects. This is useful in combination with a 'key' requirement, per the Lock section below. The players will be allowed to make a bash attempt against the door, but the game will report that the attack caused no damage.
- **Static** – If the door is static, it can not be interacted with and is treated as an inert part of the wall or building. Some like to use this flag to avoid confusing the player about which buildings they can enter.

### Lock

- **Auto-remove key** – If true, then this causes the key to vanish from the PC's inventory after it is used to open the door.
- **Close Lock DC** – The difficulty class that an Open Lock skill check must overcome to lock the door.
- **Key Required** – If true then the PC must have a specific [key](#) to open the door. Companions will not attempt to unlock a door that requires a key.
- **Key Tag** – If a key is required, then this is the tag of the key that is needed.
- **KeyRequiredFeedbackMessage** – The message to echo back to the PC when they try to open the door without the required key.
- **Lockable** – If true, then the door can be locked.
- **Locked** – If true, then the door is initially locked.
- **Open Lock DC** – The difficulty class that an Open Lock skill check must overcome to unlock the door. Typically, when a Rogue character attempts to unlock a door, the system will apply a "take 20" to the skill check. This means, for example, that a 1st-level Rogue with a 16 Dex and 4 ranks in the Open Lock skill will automatically be able to unlock a door with Open Lock DC of 27, unless they are in a combat situation.

### Misc

- **UV Scroll** – This expandable field can be used to causes the door skin to scroll in the direction and rate specified by the values of U and V, when Scroll is true.

### Scripts

These are various event handler scripts. Note that most of these events can only be triggered by a PC, and only while the PC is the controlling character.

- **On Click Script** – this is run whenever the player clicks on the door. If the door is configured as a transition, then a script in this slot will cause the transition not to occur.
- **On Closed Script** – this script runs when the door is shut.

## Doors

- On Conversation Script – *this script doesn't run, even if a conversation is launched using a script.*
- On Damaged Script – runs this script when the door is damaged.
- On Death Script – Run when the door is destroyed. By default this is set to 'x2\_door\_death', which runs a wooden explosion special effect after the door is removed.
- On Disarm Script – this script runs when a trap on the door has been disarmed.
- On Fail To Open Script – Runs this script when an open lock attempt fails to open the door. This is also run when the door is locked and the player attempts a Use action. To simulate a conversation with somebody on the other side of the door, you can use the 'gp\_talk\_door' script here. This will run the door's Conversation entry.
- On Heartbeat Script – Run this script every six seconds.
- On Lock Script – Run this script when the door is locked.
- On Melee Attacked Script – This is run each round that a bash attempt is made against the door.
- On Open Script – Runs this script when the door is opened by a Use or left-click.
- On Spell Cast At Script – this is run when a spell is cast at the door, such as *knock*.
- On Trap Triggered Script – this script runs when the door's trap is triggered.
- On Unlock Script – This script is run when the door is unlocked.
- On Used Script – *this didn't run on a right-click and select Use.*
- On User Defined Event Script – user-defined events will cause this script to run.
- Variables – This can be used to define and initialize local variables that are applicable to the door.

### Statistics

These parameters are used for door bashing attempts.

- Current Hit Points – Usually the same as Hit Points,

prior to any bash attempts.

- Fortitude Save – This is the Fort save when the door is subject to spells.
- Hardness – When the door is struck, it should absorb this much damage before losing any hit points. Unfortunately this feature does not work properly, even though the game prints out a message about how much damage was absorbed.<sup>41</sup>
- Hit Points – This is the total number of hit points that must be lost before the door is destroyed.
- Reflex Save – This is the Reflex save when the door is subject to spells.
- Will Save – This is the Willpower save when the door is subject to spells.

### Transition

These parameters allow the selection of an open doorway by the controlling PC to initiate a party transition to a new area. Note that a door that is destroyed as a result of a bash attack will still provide a party transition.

- Link Object Type – This menu allows the door to cause a transition to another door or a waypoint.
- Linked To – This is the tag of the object type selected above. When this door is opened, so too is a linked door.
- Party Transition? – If true, then the entire party will be moved to the destination. Otherwise only the PC is selected.
- Use Invisible Transitions? – *I haven't seen a difference between setting this field to true or false.*

### Trap

The following parameters apply to the door if the Trapped field is true.

- Disarm DC – This is the difficulty class that must be overcome with a Disable Device skill check in order to disarm the trap. See the skill description in the game manual for the applicability of the DC to other actions. Note that Rogues don't take 20 on a Disable Device skill check. A 1st level Rogue with a 14 Dex

---

<sup>41</sup> For a work-around, see the 'Placeable Damage Bug Fix' script in third volume.



## Doors

(+2 skill Mod.), 4 ranks in Disable Device and the Nimble Fingers feat (+2 skill Mod.) will have a 50% chance of disarming a trap with a DC of 28.

- Trap Detect DC – This is the difficulty class for finding a trap using the Search skill. Typically the Trap Detect DC should be about the same as the Disarm DC.
- Trap Detectable? – If this is false, the trap can not be detected with the Search skill.
- Trap disarmable? – If this is false, the trap can not be disarmed with the Disable Device skill. Companions will not try to disarm traps that are not disarmable.
- Trap one-shot? – If false then the trap can trigger more than once.
- Trap Type – This is a lengthy list of trap types that determine the damage and special effect when the trap is triggered. The types correspond to the traps listed in the traps.2da file, which also contains values for detecting and disarming the various traps. Only rogues can disarm these traps.
- TrapActive – The door can be trapped but inactive. The active state can be changed from a script by a call to SetTrapActive.
- Trapped – Setting this to true will cause a trap to trigger when the door is opened.
- TrapRecoverable? – If true this allows the recovery of the trap as an item on a Disable Device at DC + 10.

property of the door *A*, then the door's 'Link Object Type' menu should be set to either 'Transition to a Door' or 'Transition to a Waypoint', depending on the destination object *B*. If you want the entire party to transition with the PC, set the 'Party Transition?' property of door *A* to true.

It is often logical to configure a pair of doors to transition between each other. This is useful, for example, when creating a building entrance that corresponds to a door on an area interior. In some cases, it may make sense to place a waypoint just beyond the door and use that as the transition point rather than the actual door. This will avoid the problem of awkward camera placement following the area jump.

It may not be possible to match the style of a particular exterior door with a corresponding interior door. However, there are a few door styles that match up:

Interior Door	Exterior Door
Standard Interior Door 2	Basic Door 3
Standard Interior Door 3	Basic Door 4
Standard Interior Door 4	Basic Door 5
Standard Interior Door 5	Basic Door 6

## Transition

Active doors can serve as a transition point to a different area, which will cause the PC or even the entire party to jump to the new location. Normally such a transition door will be located on a building side in an exterior area, or a single-sided door on an interior area. (That is, the interior door does not open into another room in the same area.)

To configure a transition door, you first need to set up a destination. This can be another non-static door, or a waypoint. The destination *B* needs to be assigned a unique Tag string so that it can be identified by the transition door *A*. This tag should then be copied into the 'Linked To'

## Doors



*Here the elegant interior door '{Estate01 Door (X1)}' has been scaled down to fit into the manor house doorway.*

## Secret Doors

Although secret doors exist, they are limited in their usefulness because they will show up during the game whenever the mouse is passed over the shape, or if the 'Z' key is used. The scripting function calls don't include a method to toggle the selectability of an object, and the `CreateItem()` call can not be used to create a door so the 'gp\_secretobject\_hb' script can't be adopted for this purpose. However, see the [Concealed Passage](#) section for an alternative approach.

## Stores

A store provides an inventory of items that can be purchased by a character, as well as a place to sell previously obtained items. Stores are normally associated with NPCs and they are activated as the result of a conversation. The properties of a store are configured with a Store blueprint, and the toolset comes with a set of standard store types that can be used within a module.

The store blueprints can be accessed from the Selection panel by choosing the Stores tool from the Blueprints tab. As with other blueprints, a store blueprint can be copied and modified by selecting an existing store and right-clicking 'Copy Blueprint', or a new store blueprint can be created from the same pop-up menu.

Within an area, a store is symbolized in the toolset by a yellow arrowhead with a red flag. It is invisible during play.

## Properties

When a store is selected in the Selection panel or in an area, its properties will be displayed as a list of parameter names and values in the Properties panel under the Properties tab. The properties are subdivided into Basics, Behavior and Scripts blocks. When a selected store is in an area, a Misc block will also appear showing the location information. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

### Basics

This block is mainly used for object identification. After a store is placed in an area, several of these fields are moved to the Misc block.

- **Classification** – This determines where the store will appear in the blueprint tree. It is useful for categorizing stores that are unique to your module, such as when you want a collection a set of unique creatures for an area. Use the pipe character '|' to add sub-nodes.
- **Comment** – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing. In the toolset's stock stores, this

## Stores

is often used to recommend a character level range.

- Localized Description – This doesn't seem to be used for a store.
- Localized Name – This is the name that will appear in the blueprints Name column.
- Resource Name – Is a unique, 32-character name that serves as a blueprint identifier. It is used in the CreateObject function call.
- Tag – The string used to reference the store from a script or conversation.
- Template ResRef – The resource name of the blueprint template that this object inherits from.

### Behavior

This block determines the economic behavior of a store.

- “Will Not Buy” List – This is a list of items by Base Item type that the merchant will not purchase from the PC. Click on the ellipsis to open up a dialog that will allow you to add Base Item types to the list. Any items on this list can not be sold to the merchant; although they will still be appraised.
- “Will Only Buy” – This is a list of items by Base Item type that the merchant is willing to buy from the PC. If this list contains any items, then those items that are not on this list can not be sold to the merchant.

Note that this field and the “Will Not Buy” List parameter are mutually exclusive. If an item is on both lists, then the “Will Not Buy” takes precedence and items of that base type can not be sold. If both fields are empty then the store will buy any items.

- Black market price percentage for buying items from the player – If the Black Market flag is true, the store will buy black market goods at this percentage of the normal price.
- Black Market – When this is set to true, the store will buy stolen or black market goods from the PC. Otherwise the player will get an error message when they try to sell such an item.
- Identify price – This is the cost in gold pieces to identify a magic item for the player. A value of -1

will disable item identification. Note that the cost of an identify will be listed as 100 gp in the player's pop-up menu, but the actual cost of the item will be shown in the item description.

- Maximum Buy Price – The merchant will not spend more than this amount to purchase an item from the player.
- Price percentage for buying items from the player – This is the percentage of the full item price that the merchant will pay for an item. Normally this is set a value below 100.
- Price percentage for selling items to the player – This is the percentage of the full item price that the PC will pay for an item. Typically this is set above 100.
- Store Gold – The merchant has this total amount of gold on hand for the purchase of items.

### Scripts

These are the available event handler scripts.

- On Close Store Script – This script is run when the store is closed. It could be used, for example, to adjust the store parameters in response to the PC's purchases and sales.
- On Open Store Script – This script is run when the store is opened. It could be used, for example, to modify the store's maximum gold or identify cost based on special circumstances. The store inventory consists of item templates, so you can not apply variables to the items using a script until after they have been acquired by the PC.
- Variables – This field can be used to preset variable values for use in scripts. Select the row then click on the ellipsis to set the variable names, types and values.

In these scripts, the OBJECT\_SELF parameter is the store itself, rather than the merchant. The store parameters that can be modified by a script are the amount of gold available, the cost to identify an item, and the maximum price that will be paid for a purchase.

Unfortunately, in the versions prior to patch 1.13 it was not possible to modify the inventory of a store once it has been configured. Instead, multiple stores must be used to

## Stores

vary the inventory. A patch added the ability to view and modify the store's inventory via a script, allowing the stock to be updated over time.

### Store Tab

Once a selected store's properties have been set, you can modify the items that are available for purchase by selecting the Store tab in the Properties panel. At the bottom are tabs for the same five categories that are shown to the players: Armor, Miscellaneous, Potion, Ring and Weapon. For each of these tabs, you can click on the 'Edit...' button at the bottom of the panel to update the list.

The store editor is not selective about what items you add for each category. Thus you could, for example, choose a scroll and add it to the armor category. However, the scroll will always appear in the miscellaneous section when a character is interacting with the merchant. Thus the toolkit categories are there mainly for your convenience. (See [Lazjen's CPS Inventory Manager Plugin](#) for a tool that addresses this and other issues.)

Each item entry in the Store has an 'Infinite' check box. Selecting this box will mean the store can sell an unlimited number of these items. Otherwise it can only sell the entry one time unless it is restocked by a script.

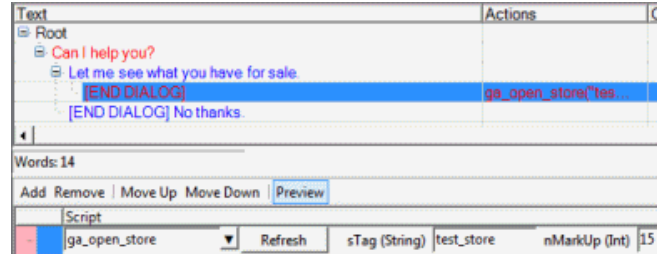
While you are modifying the Store contents, you will not be able to view the details of the items you are adding or removing. However, the Named Items chapter of the second volume can be used to gauge the cost and relevance of the magic items. You will also want to decide whether to include or exclude items that are in the store's "Will Not Buy" or "Will Only Buy" lists.

Many stores have a theme that depends on the type of business being run by the merchant. This is reflected by the standard stores types available in the toolset. For a consistent campaign setting, it may be useful to build specialized stores for particular products. Thus a leather worker could sell only leather armor, hide armor, slings, leather clothing and belts.

### Creating a Store

The addition of a store to a module involves more than simply placing a store object in an area. First, you will need to add a creature or suitable object to interact with the PC. Usually this is an NPC that is configured to serve as a shop keeper. To avoid issues with faction attitudes toward the party, it is usually a good idea to change the merchant creature's Faction ID setting to 'Merchant'.

Next, a conversation needs to be created in the A/C/S panel, then it must be inserted into the merchant creature's [Conversation](#) property field. One of the end branches of this conversation should be configured to launch the store. This is done by selecting a red [END DIALOG] entry, then choosing the Action tab at the bottom and adding the action script 'ga\_open\_store'. After clicking refresh, set the sTag field of the action to the tag of the store blueprint created earlier. This action also contains fields for marking up or marking down items, which can be set appropriately based on the conversation choices of the character or other branch conditions.



*A store 'test\_store' is opened by this conversation*

Finally, the completed store object must be placed in the same area as the merchant; by convention this is placed on the floor directly under (or near) the merchant creature. This location makes the association obvious and the store object easy to find.

When the store is opened via the 'ga\_open\_store' script in the game, an opposed Appraise check will be made between the merchant and the speaking PC. The outcome will adjust the costs up or down accordingly. (This will be further modified if charm or dominate spells are in effect.)

### Examples

In addition to the standard stores available with the toolset,

## Stores

some potential store types include:

- Artificer – Mechanism parts, trap kits, crossbows, skeleton keys and thieves tools. Will buy broken items.
- Bowyer/Fletcher – Bows, bolts and arrows.
- Exotic arms dealer – All types of exotic weapons, monk gear, cursed items, and rare items with racial or alignment restrictions. (Example: *storm armor of the Earth's children*.) Only buys items of comparable types, but may also be a black marketeer.
- Glass blowers – Empty ale stein, empty spirits bottle, empty wine bottle, magical potion bottle.
- Herbalist – Belladonna, choking powder, distilled alcohol, garlic, glands, healer kits, poison and venom.
- Instrument maker – Bandore, drum, fife, flute, lute. May have a few bard scrolls.
- Jeweler – Gems, necklaces, rings, crowns, circlets and stones. Store is well guarded.
- Leatherworker – Items primarily made of leather: armor, boots, helmets, gloves, belts, pouches, bags, scabbards, quivers, slings and some light shields.
- Library Arcanum – Recipe books and arcane scrolls. Often only buys books and scrolls.
- Money lender – Loan money for collateral and interest. May have a few special items from clients who didn't return the loan.
- Necromancy peddler – Necromantic scrolls, poison, toxins, skulls, fangs and undead remains. May also sell some necromantic scrolls, weapons and other items. Often pays well for tomb robbing.
- Outfitter – Axes, belts, boots, cloaks, daggers, gloves, light armor, shields, spears. May sell maps.
- Pub – Ale, distilled alcohol, stein, spirits and wine.
- Scrivener – Blank scrolls and (non-recipe) books.
- Tailor – Items primarily made of cloth or felt: armor, clothing, rags, robes, cloaks, hats and gloves.
- Trapper – Hides, pelts, fangs, tusks, tanglefoot bags and trap kits. Knows good hunting locales.
- Woodworker – Light wooden shields, clubs, staves, some spears and planks.

Your module could also include, for example, a black marketeer, a specialty collector, a magic auctioneer or an itinerant merchant. Specialized magic stores may only sell items created by a school of magic, such as abjuration or air elementalism. Specific races may sell magic items favored by their people. Notices may be posted announcing the sale of a magical item. Magic item bounty hunters may seek long lost relics, while also trading minor items. Some stores may require membership with an elite guild or secret group to gain access.

Note that as a campaign progresses, the player will accumulate more gold and so be able to purchase items with greater power. The availability of items in the stores will often need to be adjusted to reflect this. Early in the game, the party should usually be able to purchase alchemical supplies and various expendable items such as arrows or basic potions. At low levels, permanent items will often be rare and expensive, so they must usually be acquired by adventuring.

For realism, powerful magic items usually require that the player find an exclusive source. Thus, the PC may need to travel to a major metropolis that is home to more powerful wizards and priests, allowing expensive items to be manufactured. These sources will often be located in the wealthier parts of the city, and be well defended against thieves. Alternatively, rare items may be sold by scavengers at ancient ruins, in the relatively inaccessible abode of magically powerful beings, or by higher level NPCs.



## Placeables

### Placeables

Placeables are typically objects that represent physical structures in an area. They include buildings, curbs, walls, balconies, furniture, decorations, containers and natural features. Most placeables are inert props that serve only to occupy a volume in the game world and provide decoration, although a few are usable and can be activated or opened. Some placeables are animated, such as some of the ships and the windmills.

The placeables are sub-divided into five general categories. The 'BUILDING PROPS' are placeables for use in exterior areas. This category includes various buildings, ships and curbs. Buildings are generally inert objects, but they can have framed openings where a door can be placed.

The 'MANMADE PROPS' category contains placeables that can be used in both exterior and interior areas, and includes balconies, containers, walls, workbenches, corpses, furniture, signs, and magic portals. The scale of the man-made props ranges from a pair of dice up to wagons. Containers can have an inventory of items, allowing them to be looted or used for storage.

A third category is the 'NATURE PROPS' which can be used to add details to a natural interior or exterior area. This includes various rocks and boulders, vegetative growths, roots, wyvern nests and waterfalls. The 'Mask of the Betrayer' expansion added various snow-covered placeables that can be used in a wintry outdoor setting. 'Storm of Zehir' includes an overland map props category containing miniaturized outdoor placeables for use with the overland map. Finally there are some special purpose props that are similar to waypoints, except that they can run scripts.

### Properties

When a placeable is selected in the Selection panel or in an area, it's properties will be displayed as a list of parameter names and values in the Properties panel under the Properties tab. The properties are subdivided into Appearance, Basics, Behavior, Lock, Misc, Saving Throws, Scripts, Statistics and Trap blocks. When a selected store is in an area, the Misc block will be expanded to include the

location information. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

#### Appearance

- Appearance – This is a numerical identifier selected from a menu. It determines the placeable size, shape and texture.
- Appearance (special effect) – Some of the visual effect files can be applied to objects.
- Body Bag – *This isn't used.*
- Casts Shadows – Define the types of shadows cast by the placeable.
- Container UI Screen – *This doesn't seem to make any difference.*
- OpenState – This will determine whether a container is initially open or closed. The default state is closed.
- Receives Shadows – The types of shadows it can receive from other objects.
- Scale – This can be used to scale a placeable's dimensions. This may be useful for creating scaled furniture in communities of giants, gnomes or halflings.
- Tint – Selected placeables can have their tints adjusted. These are usually indicated by a 'TINT' in the Localized Name.

#### Basics

After a placeable is placed in an area, several of these fields are moved to the Misc block.

- Classification – This determines where the store will appear in the blueprint tree. It is useful for categorizing stores that are unique to your module, such as when you want a collection a set of unique creatures for an area. Use the pipe character '|' to add sub-nodes.
- Conversation – A [conversation](#) can be assigned to a placeable, but this must be activated from a script.
- Faction – This is Hostile by default, presumably allowing the placeable to be attacked without causing conflicts with the other factions.
- Localized Description – This is the description that

## Placeables

appears when the placeable is examined and the static field is set to false. If nothing is entered here, a default description will appear.

- Localized Name – This is the name that will appear in the blueprints Name column, and this will show on a mouse-over if the item has the 'Useable?' field set to true.
- Resource Name – Is a unique, 32-character name that serves as a blueprint identifier. It is used in the CreateObject function call.
- Tag – The string used to reference a non-static placeable from a script or conversation.
- Template Resref – The resource name of the blueprint template that this object inherits from.

### Behavior

- Blocks Line of Sight – If true then the object will block the line of sight of a creature. Line of sight is used to determine if a creature can spot enemies or objects.
- Can Talk to Non-Player-Owned Creatures? – *Unknown*.
- Default Action Preference – This selects the cursor type that is displayed when the cursor is moved over the object. The available types are: Automatic, Bash, Disable Trap, Examine and Use.
- Dynamic Collisions – If the object is non-static, setting this to true will prevent creatures from moving through or over this object. This can be combined with 'Useable?' to allow a placeable to be used but prevent creatures from moving through it.
- Has Inventory? – If the placeable is non-static and usable, then setting this to true will allow a player to access the inventory by clicking.
- Interruptible – *Unknown*.
- Inventory Size – If the 'Has Inventory?' is set to true, then this is the maximum number of inventory slots that can be used in this placeable. By default this is 136 for the containers.
- Plot – If this is true then the placeable is essential for forwarding the plot, and so it is not subject to status

effects and can not be damaged or destroyed. The players will not have a bash option when they shift-right click on the placeable.

- Receives UI Projected Textures – *This field is locked*.
- Static – If this is set to true, then the placeable is for all intents and purposes an unchangeable part of the terrain. Any inventory is inaccessible, and the object can not be destroyed or have dynamic collisions.
- Useable? – If the cursor is moved over the item then the placeable will highlight and the player will be able to perform a 'Use' action on it.
- Walkable – If true, this allows creatures to move through the placeable as though it were illusory. It is primarily for static placeables, since non-static placeables are walkable unless Dynamic Collisions is set to true.

### Lock

These fields are useful for creating locked containers.

- Auto-remove key – If true then this causes the key to vanish from the PC's inventory after it is used to open the container.
- Close Lock DC – The difficulty class that an Open Lock skill check must overcome to lock the container.
- Key Required – If true then the PC must have a specific [key](#) to open the container. Companions will not try to unlock containers that require a key.
- Key Tag – If a key is required, then this is the tag of the key that is needed.
- KeyRequiredFeedbackMessage – The message to echo back to the PC when they try to open the container without the required key.
- Lockable – If true, then the container can be locked.
- Locked – If true, then initially the container is locked.
- Open Lock DC – The difficulty class that an Open Lock skill check must overcome to unlock the container.

### Misc

- UV Scroll – If the Scroll field is set to true in this expandable entry, then the surface texture will scroll across the placeable surface at a rate determined by

## Placeables

the U and V settings. In most cases this is not useful for a placeable.

### Saving Throws

- Reflex Save –
- Will Save –

### Scripts

These are the available event handler scripts for a non-static placeable. Most of these events can only be triggered by a PC, and only while the PC is the controlling character.

- On Closed Script – run on closing the inventory.
- On Conversation Script – this is not used. To initiate a conversation with a placeable that is non-static, usable and has no inventory, you can place 'nw\_g0\_convplac' (or a comparable script) in the object's 'On Used Script' field. Doing so will use the value in the placeable's Conversation property.
- On Damaged Script – this script is run each time the placeable is successfully attacked, even if the Hardness absorbs all of the damage or the current hit points fall below zero.
- On Death Script – this is run when an object is destroyed, such as by a bash attempt or a spell.
- On Disarm Script – this event script is run if a trap is successfully disabled on the placeable.
- On Heartbeat Script – if the placeable is not static, this script will run every six seconds.
- On Inventory Disturbed Script – this script is run each time an item is added or removed from a placeable's inventory.
- On Left Click – if the placeable is usable, run this script when it is clicked.
- On Lock Script – if the placeable has the 'Lockable' property set to true, then this script will be run when the container is locked.
- On Melee Attacked Script – this is run each time a non-static placeable is attacked using a melee weapon.
- On Open Script – this runs when the container is opened.
- On Spell Cast At Script – this is run when a target-

specific spell is cast at the placeable. A spell like *magic missile* will trigger this script, while *sleep* does not. It ignores spells cast by NPCs.

- On Trap Triggered Script – this script runs when a trap is triggered on the placeable.
- On Unlock Script – this script is run when the object is manually unlocked. Typically this would be used on a container. To award XP for unlocking a door or container (perhaps one with a trap), set this to go\_xp\_lock. (The XP is three times the unlock DC.)
- On Used Script – this is run when a placeable is flagged as Useable and then is used. For a container it runs when the inventory is opened and when it is closed. See the '[On Used Scripts](#)' section below.
- On User Defined Event Script – user-defined events will cause this script to run.
- Variables – this can be used to define and initialize local variables that are applicable to the placeable.

### Statistics

If a placeable is not static and does not have the plot flag set, they it can be destroyed by a bash attempt. Any inventory will be deposited on the ground.

- Current Hit Points – Usually the same as Hit Points.
- Fortitude Save – This is the Fort save when the placeable is subject to spells.
- Hardness – When the placeable is struck, it will absorb this much damage before losing hit points. Unfortunately this does not work, even though the game prints out a message about how much damage was absorbed.<sup>42</sup>
- Hit Points – This is the total number of hit points that must be lost before the placeable is destroyed.

### Trap

These fields are useful for creating trapped containers. The following parameters apply to the placeable if the Trapped field is true. See the [Doors](#) blueprint section for a description of the fields.

---

<sup>42</sup> See the third volume for a work-around.

## Placeables

### Toolset Collections

Among the toolset placeables are several groups that share consistent visual styles. Examples include the bar pieces, bookcases, booth seating, cages, crates, jail and theater prop sets, all of which can be used for interior decoration. Objects from these groups can be located together to form various flexible combinations.

There are collections of placeables that perform a similar function:

- **Balconies** – Located in their own section (under MANMADE PROPS), these placeables can be used to create elevated traversable surfaces. There are a number of pieces available and you will need to experiment before you find the right piece for a particular location. (In general, the higher the number on the tag, the larger the balcony.) Several of the wooden 'Balcony Piece' placeables lack a back railing and are best used against the wall of a standard interior.
- **Boulders and rocks** – The NATURE PROPS section contains tintable boulder piles and rock faces. These are useful for creating natural rock fields on an outdoor area or in a cave tile, as they can be scaled to the required size and then tinted to blend into the surrounding features.
- **Bridges** – The available bridge pieces are the 'bridge' and 'dolphin bridge' in the BUILDING PROPS section. The dock pieces can also serve as a bridge, particularly the slum piece. Various linear makeshift bridges can be produced by converting the placeables to environmental objects, then covering the group with a [Walkmesh Helper](#) from the misc props.
- **Building sets** – These sets include the dock row houses, merchant row houses, Mulsantir houses, rural houses, slum row house, sunken city ruins and swamp houses. Most of the building sets are tintable, so you can give them a common color style or even use a variegated color scheme. Some of the building sets have matching copies that show burn damage, allowing a particular settlement to be transformed into a torched village with suitable visual effects.
- **Cards** – The city and merchant row house sections of the building props contain flat placeables that can be used as background filler. These should be placed in inaccessible parts of an exterior map to fill in holes along the player's line of sight.
- **Containers** – Under the MANMADE PROPS section of the blueprints is a CONTAINERS subsection with a list of containers. These are configured with their inventory field enabled and they will animate when opened or closed. To stock a container with items, you will need to place it in an area, select it and choose the Inventory tab on the Properties panel. This interface allows you to add various items and to customize their properties and appearance.  
  
Note that any of the other placeables can serve as a container. To do so, set the 'Has Inventory?' and 'Useable?' properties to true, and the 'Static?' property to false. Particular examples of container-type placeables include armor racks, bags, barrels, bookcases, cabinets, crates, the desk and liquor cabinet, potion sets, shop counters and weapon racks. However, these items will not animate in the same manner as a CONTAINER blueprint.
- **Curbs** – There are two types available in the CURBS subsection of BUILDING PROPS: flat and raised. Both types can result in impassible areas of the map following a bake, so it may help to convert them into environmental objects. However, for the raised curbs, this will result in an unrealistic effect as creatures will move across the mesh surface rather than the curb. You can address this problem by locking the curb height and position, then carefully raising the surface mesh beneath the curb using the Flatten terrain tool. However, it may prove easier to flatten the entire surface that is touching the curb, then use a small brush to lower the curb edges down to the normal surface elevation.
- **Dock pieces** – Underneath the BUILDING PROPS folder is a collection of different dock shapes that share a consistent style. However, if you attempt to link them together to form long dock sections, you will very likely experience problems during baking.

## Placeables

(See the [Exterior Areas](#) section for more details.) The best option for an extended dock is to use the 'Dock {Large}' piece and design your waterfront around it. Alternatively, a long straight dock can be covered by a [Walkmesh Helper](#).

- **Fallen trees** – There are five fallen trees in the NATURE PROPS section. You can get more mileage out of these by varying the scale, lowering them part way into the ground and partly covering them with uneven ground.
- **Floor coverings** – These are placeables with a low height setting that can be used to decorate the floor. Manmade floor coverings include: arcane circles, blankets, blood stains, dias, floors, floor mats, floorprop, plates, rugs and tarps. These are useful for breaking up the monotony of an open floor area. In the NATURE PROPS are the caves cobble piece, dirty cave floor, mines cobble piece and water planes. Typically these coverings can be converted to environmental objects. The exceptions are pallets and planks, which have some altitude so they are better left as placeables.
- **Ships** – The available vessels are the cargo ship, fishing ships, the grey ghost ship, several row boats, a ship ferry and a warship. Several ships are animated, so they appear to roll slightly in the water and are ideal for a dock scene. However, the ships are not designed to be walked upon, so any interactions with the vessel's crew will need to occur on the land or possibly in an area representing the below decks. Note that there is a cargo ship under the placeables category of the creatures blueprints, which can travel about and so be used as a wandering ship. (The underwater surface needs to be at a constant height for this to work properly.)
- **Signs** – Although some of the buildings have signs, these are inactive and can not be selected. Instead there are a variety of signs available in the MANMADE PROPS that can be placed next to establishments. By default these are static. When 'Static' is set to false, the value in the 'Localized Name' will appear during a mouse-over in the game.

You should also set 'Dynamic Collisions', 'Plot' and 'Useable' fields to true.

- **Walls** – The WALLS section under the MANMADE PROPS includes several sets of consistent wall styles. The three rural fence styles allow fences to be built on sloping terrain, and can be used for guide rails on steep paths. However, it will take careful adjustment to fit these fence pieces together. The iron fence and stone walls are useful for fencing off a private yard, while the keep wall, city wall, mulsantir wall, rural fort and thayan wall sets are designed for an outer wall around a settlement or fortification.
- **Wall Decorations** – Among the MANMADE PROPS are placeables that can be used for covering walls: banners, curtains, logram banners, masks displays, mosaic art, paintings, plaques, shelves, tapestries, torch and some lamps. When located next to a wall, these can usually be converted to environmental objects.
- **Workbenches** – These MANMADE PROPS are for use with the game's crafting rules. Each is usable and allows an inventory. The function of the workbench is determined by the object tag or a variable setting in the Scripts block of their properties.

## Crafting Items

The crafting system allows characters with the correct combination of skills, levels, inventory, feats and spells to construct new items. The workbenches can be used to manufacture magic items. These items are created according to formulae listed in recipe books that can be placed in a module for a player to discover or purchase. (Note that these recipe books do not server as triggers for constructing items; they are informational only. This information could also be provided by some other means, such as through a conversation.)

By default, the recipes for building these new items are retrieved by reading the recipe from the 'crafting.2da' file. Alternatively, the recipes can be stored in variables by running the 'ga\_setrecipes' script at some point prior to the player attempting to use them during the game, such as during a module load. If the global variable 'RecipesSet'



## Placeables

(defined in 'ginc\_crafting') is set to a non-zero value, the recipes are retrieved from memory rather than from the 2da file.

As an example, *The Book of Seeing* recipe book includes a formula for creating a *Gem of Seeing*. Internally, the recipe for this item is defined on lines 1164-65 of the 'ga\_setrecipes' file, and on row 247 of the 'crafting.2da' file. The gem's reagents list consists of the tags 'cft\_gem\_15' for the *King's Tear* gemstone and 'cft\_ess\_air2' for the *Weak Air Essence*, respectively. The `CreateWondrousRecipe()` routine is called to set the recipe for the item. The input arguments for this routine consists of the spell *true seeing*, the reagents list above, the tag X0\_IT\_MSMLMISC04 for the *Gem of Seeing*, and the minimum required level of 10.

To perform alchemy, the Mortar and Pestle item is used. This has a matching [tag-based script](#) called 'i\_mortar\_ac' that is run when the item is activated. This calls the `DoAlchemyCrafting()` routine from the 'ginc\_crafting' file, which checks for a valid Alchemy workbench. This can be any usable placeable that has the 'WB\_alchemy' variable set to a non-zero value (or any of the standard Alchemist's workbenches). If this is a workbench, a check for a valid alchemy recipe is checked against the list input with the 'ginc\_crafting' file. On a match, the ingredients are removed and the crafted item is created.

Smithing uses a similar approach, but requires the blacksmith's hammer, a basic crafting item. The matching tag-based script is 'i\_smithhammer\_ac', which calls `DoMundaneCrafting()` to implement a mundane recipe. An example of an item that can be created by this means is the Mithral Dwarven waraxe.

The magic crafting system is triggered when a spell is cast at a usable, non-static placeable with the 'ga\_forgemagic\_ca' script in the 'On Spell Cast At Script' property field. (An example of this is the Magician's Workbench placeable.) This script calls the `DoMagicCrafting()` routine in the 'ginc\_crafting' file, which checks whether there is a matching recipe for the combination of spell cast and the inventory in the placeable. If there is no such recipe, the message "Crafting failed! This is not a valid recipe" is printed. Otherwise, if the caster has the required feat and

level, the required inventory is destroyed and the enchanted item is created in its place.

In the *Mask of the Betrayer* campaign, the *Enchanter's Satchel* basic crafting item can be used to enchant an item with up to three or four item properties. The matching tag-based script is 'i\_nx1\_container01\_ci'. This runs the script 'gr\_domagiccrafting', which calls the `DoMagicCrafting()` routine described above. The types of enchantments that can be added are described in the item descriptions for the Brilliant and Pristine elemental essences. Note that this expansion use a different set of essences with different names and tags.

## Miscellaneous

The following placeables are available under the miscellaneous props (MISC PROPS) section.

### Collision Ball and Box

The Collision Ball and Collision Box are non-static, invisible objects that have their Dynamic Collisions property set to true. After a collision object is placed in an area, you can find it again by setting the 'C3 Data' field on the toolbar's Collision menu, or selecting the object from the Area Contents panel. In a game setting these objects will create invisible obstructions that will only be apparent if a player attempts to move into their location. They can be used, for example, to create barriers that can be removed by a script. (See the `DestroyObject` call in the second volume.)

An alternative use for the collision box is to make a portion of a placeable selectable, such as the hanging sign on a building. To do this, create the collision object and change the scale so that it fits just around the area to be selected.<sup>43</sup> The Interruptible, Plot and Usable properties should be turned on and the Dynamic Collisions should be turned off. The Localized Name property should be changed to the name that will appear in the game and the Localized Description to the description that will appear on an examine action by the player. You might want to change the Default Action Preference to something suitable like

---

<sup>43</sup> For example, a collision box surrounding a building's hanging sign may have a scale of 0.75, 0.1, 1.1.

## Placeables

'Examine'. Note that the collision object will not light up during a mouse-over, but the Localized Name will appear above the object.

### Ipoint

The description for the Ipoint placeable says it is used a location for placing sounds with a script. Thus, for example, a PlaySound call can be made from a script that is run from one of its Scripts properties. Alternatively an Ipoint can be referenced for its location using a GetLocation call. An Ipoint can also be used for visual effects, whether by setting the 'Appearance (special effect)' property or making an ApplyEffect\* call from a script. See the Effect Files section of the second volume for descriptions of the visual effects.

There are two variations of the Ipoint placeable:

- The 'Ipoint Cleaner' placeable is used to clean up a combat cutscene. It uses 'gp\_ipcleaner\_hb' as a heartbeat script.
- The 'Ipoint Speaker' is a placeable for running a conversation. It uses 'gp\_ipspeaker\_hb' as a heartbeat script, which calls the IPSpeakerHeartbeat routine in 'ginc\_ipspeaker'. The conversation is initiated if the PC is in the same area as the placeable and the conditions are currently safe.

### Gold Pieces

Sometimes you will want to include a stack of gold pieces in a container, such as a treasure chest. One method of doing this is to create an Item that consists of a stack of gold pieces. Typically this would consist of a round number so that you can re-use the same stack in multiple containers.

To create a stack of 100 gold pieces, first go to the Items blueprints and create a new module blueprint. This item should have the following properties:

- *Icon:* it\_gold
- *Base Item:* Gold Piece
- *Classification:* Miscellaneous|Gold
- *Localized name:* 100 Gold Pieces
- *Quantity:* 100

When editing a container inventory, you can now add this item to provide 100 gold pieces when selected by the

player. Creating items with multiple gold piece amounts will allow you to provide different allotments.

### Lever, Useable

This is a placeable that provides an operational lever. For this object to perform a useful action, a script should be provided in the 'On Used Script' field. The lever can be animated with an ActionPlayAnimation call in this same script. For example:

```
void main() {
    // Animate the lever
    ActionDoCommand( ActionPlayAnimation(
        ANIMATION_PLACEABLE_ACTIVATE,
        1.0f, 0.5f ) );
    PlaySound( "as_sw_lever1" );
    DelayCommand( 0.5f, ActionPlayAnimation(
        ANIMATION_PLACEABLE_DEACTIVATE
        1.0f, 0.5f ) );

    // Execute "LeverAction" script, if set
    string sLeverScript = GetLocalizedString(
        OBJECT_SELF, "LeverAction" );
    if ( sLeverScript != "" )
        ExecuteScript( sLeverScript,
            OBJECT_SELF );
}
```

At the end of the call, a check is made for a string variable named "LeverAction". If this variable set on the lever, activating the lever will cause the script with the matching name to be executed. The Local Description property should also be modified to provide a description of the lever. This will appear if the player performs an Examine on the lever in the game.

### Secret Object

The Secret Object is a special purpose, invisible placeable that can be used to reveal a hidden object when a character makes a successful search skill check. This is implemented through a heartbeat script 'gp\_secretobject\_hb' and a set of local variables on the object. First, this script checks to see if a search is successful, then it creates a new object at the location of the placeable.

Here are the variables that can be set on the Secret Object:

## Placeables

- SearchRadius – This is the maximum search radius. The creature must also be within the line of sight of the placeable location.
- SearchDC – The difficulty class of the Search skill check.
- SearchDCDetectMode – This is the difficulty class when the creature has Detect Mode active. Elves automatically use this mode.
- SecretObject\_ResRef – The template resource reference string of the hidden object to create upon discovery. This can be a creature, item or placeable, but it can not be a secret door.
- NewObjectTag – The tag string to give the newly created object.
- VisualFX – This is the ID of the temporary visual effect to apply to the revealed object for 10 seconds.
- ReplacedObjectTag – This is the tag string of an object to destroy when the search is successful.
- JumpDestinationTag – *This appears to be intended for secret doors.*
- TimeToReset – If this is non zero, the objects will be put back the way they were and the placeable will reset itself.
- RunScript – If this is not a null string, the script matching this name is run when the hidden object has been revealed.
- Class\_Restriction – *This is not used by the script.*

### Concealed Passage

Using the 'Secret Object' placeable above, it is possible to create a secret passage that will not show up on the players map and may not be immediately apparent within the game. This works especially well with the caves tile set.

- Begin with a doorway in an interior area. On the other side of this door place a two-door room tile, such as 'Door{2\_Door\_Room\_Variant}' and place a closed door in the second opening. (The original doorway should be left open.)
- In the 'ESTATE TILESET' is a 'Lid {Estate Tileset 01 (X1)}' placeable. This can be used to hide the small room. The scale of the lid needs to be changed to '3.1,

3.1, 1' and the placeable should be situated over the room so that it is completely concealed. Once you are satisfied with the placement, modify the last field of the 'Position No Snap' property to a value of 50 and set the 'Height Lock' to true. Set 'Static' and 'Walkable' to *true* or else the passage will be impassible.

- The 'WALLS' category of the 'MANMADE PROPS' contains a '{TileBlock 01}'. A copy of this placeable should be placed so as to fill the room, but not poke through into the player's starting area. Set 'Static' and 'Walkable' to *false* and give it a unique tag.
- Conceal the outer doorway opening with one or more suitable placeables. The 'Walkable' properties on these placeables should be set to *true*.

In the caves tile set, you can use one of the tintable Rock Face placeables and scale it to about 22% of normal. Once the tint matches the cave tile hue, it should be virtually indistinguishable from the surrounding rock face.

For a standard interior, a pair of bookcases may serve here, such as 'Bookcase {013 – TINT}' and 'Bookcase {014 – TINT}'. But you will need to increase the vertical scale on these objects to cover the door.

- Place a 'Secret Object' next to the opening. Set the 'ReplacedObjectTag' variable to the tile block tag.

Now the Secret Object will check for discovery and will remove the TileBlock placeable on success, thereby making the concealed room visible within the game. Unfortunately, the Lid will prevent the room from ever showing up on the map, but that is unavoidable.

### Walkmesh Helper

These placeables provide an invisible walking surface that can be used to bridge a gap in the floor. They can also be used as traversable surfaces to span elevated placeables, such as multiple bridge or dock sections that have been changed into environmental objects. The difference between the two built-in walkmesh helpers is the sound made when a creature moves across the surface. You have a choice of wood or stone footstep sounds.

## Placeables

The size of the walkmesh helper is modified with the Scale property. You will need to try different values to find the rectangle that is suitable for a particular purpose. The rectangular surface outline of a walkmesh helper can be modified with the [walkmesh cutter](#) in the Triggers blueprints. Each region where you place a [walkmesh cutter](#) will be act as a blocking object, preventing creatures from entering. Thus, for example, if the walkmesh helper is covering an irregular surface, such as a series of boulders spanning a river, then you can use the [walkmesh cutter](#) to remove the areas where the PCs are not allowed to enter.

You may need to experiment with the placement of a walkmesh helper to make it bake properly. If the object is placed too far above the surface, a walkmesh helper area can result in unnatural changes in a PC's vertical position.

### On Used Scripts

The toolkit includes several custom scripts that can be used in combination with placeables. These scripts begin with a prefix of 'gp\_' and they are typically placed in the 'On Used Script' field of a placeable that has the 'Useable?' property set to true.

- `gp_plc_to_item_us` – This script causes the associated object to self-destruct, then places an item in the inventory of the creature that executed the use action. The placeable's sRR local string variable contains the template of the item that will be created. Thus, for example, the 'Armor Pile {01 (X1)}' placeable could have a sRR variable set to 'nw\_aarcl011', causing it to become banded mail when the placeable is used.
- `gp_rotate_us` – This will rotate the placeable by an amount determined by local variables on the object. Thus this script could be used to create a rotatable statue that turns by a fixed angle and makes a scraping noise each time it is moved. Multiple rotatable statues could be used as parts of a puzzle, for example. The required variables are listed in the notes for this script.
- `gp_talk_object` – This script will cause a placeable to hold a conversation with the player using the entry in the placeable's [Conversation](#) slot. The conversation

can then include actions such as a `ga_lock` script that will change the lock status of a door.

- `gp_treasure_op_de` – This script will generate random treasure based upon local variable settings on the container. The valid values for these variables are listed in the 'x2\_inc\_treasure' include file. The "TreasureClass" integer variable determines whether the treasure is of low, medium or high value. The "TreasureType" variable is a set of bit flags that determine the treasure types.

## Triggers

### Triggers

A trigger is an object that is assigned to a polygonal region on an area map. When a PC enters the trigger region, the trigger object will activate and run a script or launch an event. Triggers can be used to activate a map note, start a conversation, unleash a trap, save the game, cause an area transition, apply effects, and so forth. The particular function that a trigger serves will depend on the blueprint's properties. Many of the built-in triggers execute a script from the toolset library.

A trigger object is created by selecting the blueprint in the Selection panel. When the cursor is moved over an area in the Edit panel, it will change to a crosshair. At this point you can select consecutive corners of the trigger region by left-clicking in the area. The region boundaries will appear as a sequence of yellow lines, forming a polygon. At least three points must be chosen to make a valid trigger region.

Typically the trigger should be placed and sized so that the party can not bypass it by skirting around the edges. Thus a map note trigger should cover all of the possible approaches to the identified location. (Alternatively, multiple copies of the trigger can be placed, each of which will activate the map note from a different region.) Once a trigger region has been defined in an area, press Esc to complete the edit.

With an area open in the Edit panel, you can select a trigger region as an object and configure its properties appropriately. Within the game, if a script is entered into the 'On Enter Script' parameter, then when a member of the party enters the active trigger region, the trigger will execute that script. Not every trigger is activated by entry into the region; for example, the 'World Map Transition' trigger will activate upon clicking the floating map.

### Properties

When a trigger is selected in the Selection panel or a placeable region in an area, its properties will be displayed as a list of parameter names and values in the Properties panel under the Properties tab. The properties of a trigger are subdivided into Appearance, Basics, Basicssarm, Lock, Scripts, Transition and Trap blocks. When a trigger region

is selected in an area, an additional Misc block will be inserted that will include location information. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

#### Appearance

- Load Screen – *Not used.*
- Mouse Cursor – This doesn't work. There is no cursor shape change when the cursor is over the trigger.

#### Basics

After a trigger is placed in an area, several of these fields are moved to the Misc block.

- Classification – This determines where the trigger will appear in the blueprint tree. It is useful for categorizing triggers that are unique to your module. Use the pipe character '|' to add sub-nodes.
- Comment – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing.
- Faction – This is set to 1.
- Localized Description – This is not used.
- Localized Name – This is the name that will appear in the Name columns for Area Contents and Blueprints.
- Resource Name – Is a unique, 32-character name that serves as a blueprint identifier.
- Tag – The string used to reference a trigger from a script or conversation.

#### Lock

- Auto-remove key? – *Not used.*

#### Scripts

These are the event handler scripts. Many of these events can only be triggered by a PC, and only while the PC is the controlling character.

- On Click Script – This script should be activated when the player left-clicks in the trigger region, but I haven't been able to make it function in most instances. The exceptions are area transition triggers such as 'World Map Transition', but these will always paint a floating map image.<sup>44</sup> The clicking object is

---

<sup>44</sup> An alternative is to use a placeable. You could, for



## Triggers

returned by `GetClickingObject()`.

- On Disarm Script – this event script is run if a trap is successfully disabled.
- On Enter Script – This script is fired when the trigger region is entered. The `GetEnteringObject()` call will return the object that triggered this script.
- On Exit Script – This runs when the creature leaves the encounter trigger region. `GetExitingObject()` will return the object that exited the region.
- On Heartbeat Script – This runs every six seconds. The `GetFirstInPersistentObject()` call can be used to find the first creature in the trigger region, followed by repeated calls to `GetNextInPersistentObject()` to locate the other targets. These should be used sparingly, however, as detailed heartbeat scripts can impact game performance.
- On Trap Triggered Script – If the 'Trigger Type' is set to Trap, then this script is run when the trap is triggered.
- On User Defined Event Script – user-defined events will cause this script to run.
- Variables – This is used to define and initialize local variables for the trigger.

### Transition

- Link Object Type – By default this is set to 'No transition' It can be changed to transition to a door or a waypoint. This will cause the creature entering the trigger region to transit to the corresponding object identified by the 'Linked to' field.
- Linked to – This can be set to the tag of a door or waypoint that the PC will transition to depending on the setting of the 'Link Object Type' field above.
- Party Transition? – If true then the entire party will be transited, rather than just the entering creature.

### Trap

See the [Doors](#) blueprint section for a description of the fields.

---

example, set a Walkmesh Helper's 'Static' parameter to false, set 'Usable?' to true and add a script to 'On Left Click'.

## Blueprints

Here is a summary of some of the available trigger blueprints.

- Autosave Trigger – When a creature enters this trigger region, it runs the `gtr_autosave` script, which will perform an autosave for a single-player game.
- Block Trigger – This runs the `gtr_block_trigger` script on entry. If the waypoint defined by the trigger's Waypoint variable is a valid waypoint in the area, the screen will fade to black, the party will jump to the waypoint, and the view will fade back from black. Optionally, a single line conversation belonging to the trigger can then be played.
- Enable Map Note – Upon entering the trigger region, this will run the `gtr_enable_map_note` script. This enables a map note, which should be a Disabled Map Note waypoint. The variable `MAP_NOTE_TAG` should be set to the tag of the waypoint.
- Henchman Trigger – The blueprint comments say that this only works with the XP1 henchman event system. *I haven't delved into this component yet. More information is needed.*
- Murder Trigger – This triggers the `gtr_murder_en` script on entry. The `sCreatureToMurder` variable is set to the tag of a creature, and the script applies a death effect to that creature.
- New Area Transition – This will create a world map transition symbol. This could be used to run a script to execute the `ShowWorldMap` call, displaying the world map.
- New Generic – By default this does nothing.
- PC-Only Area Transition – This will create a world map transition symbol. When a PC clicks the symbol, it will run `gtr_pc_tran clic`. The destination is set by the parameters in the Transition block of the trigger properties.
- SpeakTrigger – Upon entering the trigger, the script `gtr_speak_node` will be executed. This will initiate a conversation with the NPC that has the tag matching the `NPC_Tag` variable. The conversation can consists

## Triggers

of a single string which will float above the NPC, or it can be a full cut-scene conversation.

- **Tracks Trigger** – Upon entry, this trigger will run the script `x1_tracks_trig`, causing the trigger to speak a one-line conversation using the resource reference string that matches the trigger's tag string.
- **Walkmesh Cutter** – This trigger region becomes unwalkable when the area is baked. It can be used in combination with the Walkmesh Helper placeables, or for blocking off a region occupied by many environmental objects, such as trees and shrubs. Note that it is better to keep the region of the walkmesh cutter relatively simple, as a complex unwalkable area may result in a PC entering a location and being unable to escape. Blocking off an region filled with many objects using a rough outline is preferable to delimiting each object with an elaborate walkmesh cutter region.<sup>45</sup>
- **World Map Transition** – This will create a world map transition symbol. When a PC clicks the symbol, it will run `'gtr_world_map_cl'`. If the party is gathered together, it will call the `DoShowWorldMap` function in the `ginc_world_map` include file. The parameters that are passed to this call are the values of the `sMap` and `sOrigin` variables.
- **X0\_SAFEREST** – This trigger is used to identify rooms where the party can safely rest. For details, see the [Allow Limited Rest](#) section.

## Encounters

---

These blueprints provide a trigger that can bring into play a predefined group of creatures. An encounter consists of a trigger region in an area, along with one or more spawn points. When a trigger is activated by a PC or creature entering the region, the encounter's creatures will be inserted into the game environment at the spawn points. The properties of an encounter are configured by the blueprint settings. The toolset comes with a set of standard encounters that can be used within any module.

When an area is opened for editing and an encounter blueprint is selected, the cursor changes to a crosshair on the Edit panel. The encounter region can now be created by successively selecting three or more corners, in the same manner as a trigger, then pressing the 'Esc' key when the region is complete. Spawn points can be added to the encounter by selecting the region as an object, selecting the 'Paint Spawn Point' option from the toolbar, then left-clicking in the area. Press 'Esc' when you are done.

As long as an encounter region is selected, the associated spawn points will show a red flag; otherwise the flags are yellow. With a region selected, it's spawn points can be selected and relocated or deleted. If neither the encounter region nor a spawn point is currently selected, then none of spawn points are selectable. Deleting an encounter region will also delete the associated spawn points.

For realism, the spawn points should be placed so that the appearance of the spawned creatures seems natural. This can be accomplished by placing spawn points at a suitable distance from the trigger region, or at locations adjacent to concealed sites. The placement of the spawn points can also be used to create interesting tactical challenges for the players, such as a surprise attack from a flank in combination with the main attack from the front.

## Properties

When an encounter is clicked in the Selection panel or a placeable region in an area, it's properties will be displayed as a list of parameter names and values in the Properties

---

<sup>45</sup> In many cases it may be helpful to bake a group of adjacent objects so as to show their path blocking outline when the Baked flag is set, then convert the group to environmental objects and apply a Walkmesh Cutter along their outline.

## Encounters

panel under the Properties tab. The properties are subdivided into Basics, Behavior and Scripts blocks. When an encounter trigger region is in an area, a Misc block will be inserted that will include the location information. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

### Basics

After an encounter is placed in an area, several of these fields are moved to the Misc block.

- **Classification** – This field determines where the encounter will appear in the blueprint tree. It is useful for categorizing encounters that are unique to your module. The standard encounters are classified according to difficulty, and range from Easy to Very Hard. Use the pipe character '|' to add sub-nodes.
- **Comment** – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing.
- **Faction ID** – If the creature entering the encounter trigger region is hostile toward this faction ID then the encounter will trigger. Thus a creature belonging to the Hostile faction will not trigger an encounter that has a Faction ID of Hostile.
- **Localized Description** – As an encounter is not a viewable object, this has no effect.
- **Localized Name** – The name that appears in the Blueprints and Area Contents lists. This is used as a summary of the encounter type.
- **Resource Name** – Is a unique, 32-character name that serves as a blueprint identifier. It is used in the CreateObject function call.
- **Tag** – The string used to reference the encounter from a script or conversation.
- **Template ResRef** – The resource name of the blueprint template that this object inherits from.

### Behavior

- **Active** – If this is true then the encounter can be triggered. If false, then the encounter must be set to active using a SetEncounterActive call from a script.
- **Auto-reset Count** – If 'Auto-reset?' is true, this sets

the maximum number of times that the encounter can be reset.

- **Auto-reset Delay** – If 'Auto-reset?' is true, the encounter will reset after this number of seconds.
- **Auto-reset** – If this is set to true, then the encounter will reset itself to active after being triggered.
- **Creature Spawn Style** – This menu determines how the creatures will be distributed among the possible spawn points. The default is to place the monsters at a randomly chosen spawn point.
- **Creatures** – Selecting the ellipsis on this entry will bring up a edit dialog that can be used to choose the mixture of creatures in the encounter. Each row consists of a creature blueprint, a minimum and maximum number of creatures, and whether it is a single spawn.
- **Difficulty** – This menu is used to set the difficulty level of the encounter. The possible choices are Very Easy (1/2), Easy (1), Normal (2-4), Hard (5-9) and Very Hard (10-20).
- **Maximum Number of Creatures** – This is the upper limit on the creatures that the encounter will spawn.
- **Minimum Number of Creatures** – The lower limit on spawned creatures.
- **Player Only** – If true, then the encounter can only be triggered by a PC.
- **Spawn once?** – If false then the encounter will spawn additional creatures as previously spawned creatures are killed.

### Scripts

These are the available event handler scripts that are specific to encounters. Many of these events can only be triggered by a PC, and only while the PC is the controlling character.

- **On Entered Script** – This script is fired when the encounter trigger region is entered. It is run even if the encounter is inactive.
- **On Exhausted Script** – This script should be run when all the creatures spawned by an encounter are killed. *I've run encounters where this didn't fire at the end.*

## Encounters

- On Exit Script – This runs when the creature leaves the encounter trigger region. It is run even if the encounter is inactive.
- On Heartbeat Script – This is run once every six seconds.
- On User Defined Event Script – This script is run upon the triggering of a user-defined event.
- Variables – This is used to define and initialize local variables for the encounter.

## Sounds

---

The blueprints in the Sound category contain a number of auditory effects that can be used to produce localized sounds in an area. The sounds are generated from '.wav' files in the game's install folder, although custom sounds can also be used from your local 'override' folder. When placed within an area, a sound object is shaped like a blue speaker with a hexagonal cone.

When the 'Placed' option in the toolbar's 'Sound' menu is active, the output from placed sound objects can be heard within the toolset. For a sound object to be audible, the frustrum within the Edit panel must be positioned within the Maximum Distance radius of the sound. The maximum radius of a placed sound is shown by a sphere of tiny purple dots.

The toolset sound blueprints are organized into the following categories:

- Background – This is a mixture of generic sounds found in various settings, such as tools being use, building noises, distant weather and the movement of vegetation.
- Chatter – This is for conversations and noises produced by men, women and children in various circumstances, ranging from taverns to contests. It includes the Walla sub-category, which is the general noise produced by groups of people.
- City & Town – The subcategories consist of bells, docks, rope use, smithy and wood sounds. The last is useful for creaking old wooden buildings.
- Creatures – These sounds reproduce the noises emitted by specific creatures, such as dragon, ghost or orc.
- Dungeon – The natural sounds heard in caves and mines.
- Environmental – These are elemental-type sounds divided into fire, lava, steam, water and wind. The last two are the most useful for outdoor weather scenes. Fire is useful for various sources of flame, such as torches or fireplaces.

## Sounds

- **Magic** – A selection of odd sounds that can be used in a magical-theme environment, such as a witch's cave.
- **Special** – A catch-all category for sounds that don't fall into the above classes. They consist of falling body sounds, theme sounds for entrances, and a couple of musical instrument sounds.

Properly used, sounds can bring an area to life. Localized sounds can bring an environment to life by highlighting points of interest as the player explores an area. There are a number of sounds available, and many can be matched up with the nearby environment. Complete silence can seem unnatural and so should be rare. Even an empty building at night can creak and groan from time to time. The volume of localized sounds should be set to a level where they can be readily heard by a nearby player.

The available blueprint sounds differ from the list of sounds that can be set for the ambient sounds in an area. Typically the ambient sounds would be set for global audio effects set at a lower volume. The blueprint sounds are localized audio effects that are related to specific environmental features. Thus, "Tavern, Rowdy" could be used for general tavern background noise, while "Tavern Barmaid" would be used where the barmaid is serving drinks.

### Properties

When a sound is selected in the Selection panel or a placeable region in an area, its properties will be displayed as a list of parameter names and values in the Properties panel under the Properties tab. The Sound properties are subdivided into Basics, Behavior, Position and Scripts blocks. When a sound object is placed in an area, a Misc block will be inserted that will include the location information. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

#### Basics

After a sound is placed in an area, several of these fields are moved to the Misc block.

- **Classification** – This determines where the sound will appear in the blueprint tree. Use the pipe character "|"

to add sub-nodes.

- **Comment** – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing.
- **Localized Description** – As a placed sound is not a viewable object, this has no effect.
- **Localized Name** – The name that appears in the Blueprints and Area Contents lists. It is typically used as a summary of the sound type.
- **Resource Name** – Is a unique, 32-character name that serves as a blueprint identifier.
- **Tag** – The string used to reference the sound from a script or conversation. It is used in the SoundObject\* function calls.
- **Template ResRef** – The resource name of the blueprint template that this object inherits from.

### Behavior

- **Active?** – If this is false, then the sound must be activated through a script. An inactive sound can be activated with a SoundObjectPlay() call, then turned off by SoundObjectStop(). If this field is true, then the sound will continue to play even if a script calls SoundObjectStop().
- **Continuous?** – When this is set to true and the 'Looping' property is false, the sound will repeat after each time period set by the 'Interval' field, after accounting to the 'Interval Variation'.
- **Hours at which the sound is active** – If the 'Times at which to play the sounds' property is set to 'Use Hours', this boolean array determines what hours of the day the sound will be active.
- **Interval** – If 'Continuous?' is set to true, then this value is the average time interval in milliseconds before the sound is repeated.
- **Interval Variation** – If this is not zero, the 'Interval' above will be randomly modified to a value that lies between (Interval – Variation) and (Interval + Variation).
- **Looping** – If this is true, the sound endlessly repeats. A slight pause may be heard between each loop.



## Sounds

Some of the sounds are set to looping, but may be better used with the 'Continuous?' property set to true.

- **Pitch Variation** – This is a decimal value between 0 and 1. Each time the sound repeats, the pitch is randomly modified by this proportion. Thus, if the pitch variation is 0.3, the pitch varies from 70% to 130% of the base pitch. Variation in pitch helps to keep the sounds from feeling repetitive.
- **Priority** – When multiple sounds occur at the same time, the game engine assigns each a priority based on this setting. The ranking of the priorities is listed in the 'prioritygroups.2da' file.
- **Random?** – If this is set to true and there are multiple sound files in the 'Sounds' field, the sound file will be randomly selected during each loop.
- **Sounds** – This field contains one or more '.wav' file prefixes. Clicking on the ellipsis will launch a dialog where you can add sound entries, select a value and set the Sound field to the file prefix. Valid sound files will be played in the order listed unless 'Random?' is set to true.
- **Times at which to play the sounds** – This controls the hours of the day when the sound can be played. The available choices are Always, Day, Night and Use Hours. When the game clock is in a time interval when a sound will not play, then no sound will be produced even with a `SoundObjectPlay()` call.
- **Volume** – This slider sets a relative sound volume between 0 and 127, where zero is off and 127 is the maximum volume.<sup>46</sup>
- **Volume Variation** – If this is not zero, the volume will randomly increase or decrease by up to this amount each time the sound plays.
- **Elevation** – The sound will radiate from the height of the object plus this offset.
- **Maximum Distance** – If 'Positional?' is true, this is the maximum range that the sound will be audible. The volume will increase with decreasing distance up to the Minimum Distance from the sound object. This

radius is graphically displayed in the toolset by a sphere of violet dots.

- **Minimum Distance** – Within this radius the sound plays at its maximum volume. If you want the sound to be clearly audible within a certain radius, you should increase this value to a significant fraction of the Maximum Distance.
- **Positional** – If this is true, then the sound volume varies based on the distance parameters above. If it is false then the sound behaves like an ambient noise that is heard at the same volume throughout the area. The latter is useful when you want to use multiple overlapping ambient sounds within an area.
- **Random Position?** – If this is true, then the position of the sound varies randomly each time it plays. This could be used, for example, with some environmental background sounds that can change position, such as Twig Snaps or Brick Scrapes.
- **Random Range (x)** – This is the random offset along the x-axis that is applied to the position if 'Random Position?' is true.
- **Random Range (y)** – Similar to the above, except it is applied along the y-axis.

### Scripts

- **Variables** – This is used to define and initialize local variables for the sound. It can be used, for example, to track when a sound is playing because of a script.

You can make your own sound objects by copying a '.wav' file in 'override' in your NWN2 documents folder, then creating a new sound blueprint and setting the Sounds field to include the file prefix. Thus a file called 'crunch.wav' will have 'crunch' entered in the Sounds field. You will still need to set the various other fields, such as Volume, Priority and Active?.

---

<sup>46</sup> The range [0, 127] is used because this is the range allowed by an 8-bit signed character.

## Waypoints

### Waypoints

A waypoint marker provides a location within an area that can be used as a party destination, a point of interest for a map, one of a sequence of positions to march a creature, a place to spawn creatures, or a location to dynamically place an object or effect. Waypoints are not visible within the game, but they can be referenced by scripts.

### Properties

When a waypoint is selected in the Selection panel or in an area, its properties will be displayed as a list of parameter names and values in the Properties panel under the Properties tab. The properties are subdivided into Appearance, Basics, Behavior and Scripts blocks. When a waypoint trigger region is in an area, a Misc block will be inserted that will include the location information. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

#### Appearance

- **Color** – This sets the color of the flag on the waypoint marker. In addition, when a waypoint has a map note, this will be used as the color of the icon on the map.

#### Basics

After a waypoint is placed in an area, several of these fields are moved to the Misc block.

- **Classification** – This determines where the waypoint will appear in the blueprint tree. It is useful for categorizing waypoints that are unique to your module. Use the pipe character '|' to add sub-nodes.
- **Comment** – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing.
- **Localized Description** – The waypoint can not be seen by the player, so this is unused.
- **Localized Name** – The waypoint can not be seen by the player, so this is only used in the blueprints list.
- **Resource Name** – Is a unique, 32-character name that serves as a blueprint identifier.
- **Tag** – The string used to reference the waypoint from

a script or conversation.

- **Template ResRef** – The resource name of the blueprint template that this object inherits from.

#### Behavior

These fields are used to configure a note to appear on the player's map.

- **Has Map Note?** – If this is set to true, then a map note can appear on the player's map at this location.
- **Map Note Enabled?** – When a map note is enabled, it will appear on the player's map. If it is disabled, a trigger script can be used to enable the note.
- **Map Note Text** – This is the string that will appear on the player's map when a map note is enabled.

#### Scripts

- **Variables** – This is used to define and initialize local variables for the waypoint.

### Types

The following list briefly describes some of the waypoints that are available in the toolset.

- **Block Waypoint** – Intended for use with the Block Trigger.
- **Disabled Map Note** – This is configured as a place to show a note on the player's map. The note is disabled but can be enabled by a 'Enable Map Note' trigger.
- **Map Note** – This is configured to show a note on the player's map. The text in the 'Map Note Text' field will be displayed during a mouse-over on the map.
- **Observation Point** – Marks a point of interest.
- **Point of Interest** – Marks a point of interest.
- **Rest-Encounter Spawn Point** – This was used in the *Hordes of the Undermountain* game as a creature spawn point for the rest system, as defined in the 'x2\_inc\_restsys' include file.
- **Shopping Point** – Marks a [Store](#) location.
- **Waypoint** – Generic, general purpose waypoint.

#### Ambient Animations

The default 'On Spawn In Script' value for most creature

## Waypoints

blueprints is `nw_c2_default9`. This script will check the creature for several local integer variables with names that are defined in the `x2_inc_switches` include file. Setting these to 1 will activate specific spawn-in conditions. In particular, setting `'X2_L_SPAWN_USE_AMBIENT'` equal to 1 will cause the creature to execute ambient animations, while `'X2_L_SPAWN_USE_AMBIENT_IMMOBILE'` will cause it to run non-mobile ambient animations.

The ambient animations are executed by the standard `'nw_c2_default1'` heartbeat script when it calls the `PlayMobileAmbientAnimations` function. This routine is found in the `'nw_i0_generic'` include file.

Several waypoints are available in the toolset for use with the ambient animation scripts.<sup>47</sup>

Ambient Animation Waypoint	Tag
Detect Mode Toggle	NW_DETECT
Generic Stop Waypoint	NW_STOP
Home Waypoint	NW_HOME
Safe Waypoint	NW_SAFE
Shop Waypoint	NW_SHOP
Stealth Mode Toggle Waypoint	NW_STEALTH
Tavern Waypoint	NW_TAVERN

Typically a creature's mobile animation will cause it to move to a random object or waypoint that has the tag `NW_STOP`. If a creature is spawned in an area containing the `NW_HOME` tag, the creature records that area as it's home and returns there at night. If the creature is spawned with the `inanimate animations` flag, it will stay close to the nearest `NW_HOME` waypoint, but will interact with its environment. Creatures that are spawned outdoors go through doors that lead to areas with the `NW_TAVERN` or `NW_SHOP` tags, then come back out later.

If a creature is moving ambiently and the nearest waypoint has the `NW_DETECT` tag, then it will trigger detect mode. If the nearest waypoint has the `NW_STEALTH` tag, then the creature will trigger stealth mode. When a creature suffers damage, it will move to the nearest waypoint with the tag `NW_SAFE`, then rest to recover its health.

---

<sup>47</sup> See the `PlayMobileAmbientAnimations_NonAvian` function comments in the `"x0_i0_anims"` include file.

### Map Notes

A map note appears on the player's map as a diamond shape, and moving the cursor over the note will display the map note message with a pop-up. Each map note is created using a waypoint object. In the Behavior section of the waypoint's properties, set the 'Has Map Note?' flag to True, then modify the 'Map Note Text' field to the note you want to appear on the map. If you initially want to hide the note, set the 'Map Note Enabled?' flag to false. In this case, a trigger can be used to activate the map note.

If there is not an already existing waypoint (such as a transition point), a copy of the 'Map Note' waypoint blueprint will serve. There is a corresponding 'Enable Map Note' trigger blueprint that can be used as a map note enabler. In the Scripts section of this trigger is a variable called `MAP_NOTE_TAG`. This should be set to the tag of the map note waypoint that you want to enable.

### Post

The 'Post' waypoint can be used for a guard post. After creating an NPC guard creature, place a Post waypoint at the location where it's watch will be held. The tag of this waypoint should be set to the string `"POST_"` plus the tag of the guard.

The `WalkWayPoints()` call needs to be included in the script placed in the 'On Spawn In Script' slot. The default `nw_c2_default9` script will make this call for you. This function will cause the guard to return to its post after combat.

### Walk Path

The standard `nw_c2_default9` script configures a spawned creature to follow a set of waypoints. These must use a tag of the form `WP_NPC_TAG_##`, where `NPC_TAG` is the tag of the spawned creature and `##` is one of a sequence of integers beginning with '01'.<sup>48</sup> Thus a creature with the tag `'my_creature'` will follow a sequence of waypoints with the following tags:

---

<sup>48</sup> If a single `WP_NPC_TAG_01` waypoint is created in an area, then the character will keep attempting to return to the position of that waypoint. This is useful, for example, when you want to keep a shopkeeper near a desk.

## Waypoints

```
WP_my_creature_01  
WP_my_creature_02  
WP_my_creature_03  
...
```

When a sequence of walk path waypoints is placed in an area, the toolset will automatically link them together, in numerical sequence, by a series of white arrows. To create such a sequence, place the first waypoint and assign it a tag, then duplicate the waypoint and increment the tag suffix on each copy. The white arrows should automatically appear. You can use these arrows to check that the walk path does not intersect obstructions that may disrupt movement.

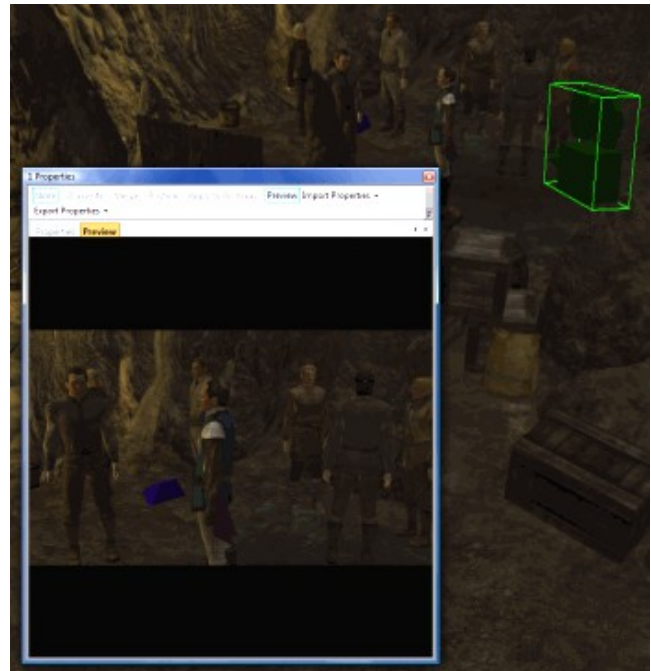
You can further enhance the behavior of a creature following a walk path through the use of scripted waypoints. This allows a script to be executed each time a creature reaches a waypoint along the walk path. The default name of this script is `WP_NPC_TAG`; thus matching the prefix used for the walk path waypoint tags.

In a scripted waypoint script, the `GetCurrentWaypoint` routine from the "x0\_i0\_walkway" include file can be used to determine the waypoint number in the sequence that the creature reached. The "ginc\_wp" include file also has a number of routines that are useful for a scripted waypoint script, such as conversation routines. See the [Writing Scripts](#) section for more information on how to write scripts.

## Static Cameras

These blueprints can create motionless viewpoints for use in conversation cut scenes.<sup>49</sup> Within the toolset, a Static Camera has the appearance of an old-fashioned film movie camera. (This shape will not be visible within the game.) When this camera object is selected in an area, you can hold down both Ctrl and right click, then drag to orient the camera in whatever direction you choose.

For better control over the camera aim, select the Preview tab in the Properties panel. You will be shown the view from the perspective of the camera. However, you have no control over the view dimensions – it is fixed to the standard size used in the cut scene.



*The view from the camera (selected at upper right) is previewed in its properties window.*

## Properties

When a static camera is selected in the Selection panel or in an area, its properties will be displayed as a list of

<sup>49</sup> See the '[Camera Shots](#)' section of the chapter on [Conversations](#) for details on how to include a static camera view during a conversation.

## Static Cameras

parameter names and values in the Properties panel under the Properties tab. The properties are subdivided into Basics and Scripts blocks. When a static camera is in an area, a Misc block will be inserted that will include the location information, as well as the pitch and roll of the camera. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

### Basics

After a static camera is placed in an area, several of these fields are moved to the Misc block.

- **Classification** – This determines where the static camera will appear in the blueprint tree. Use the pipe character '|' to add sub-nodes.
- **Comment** – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing.
- **Localized Description** – As a static camera is not a viewable object, this has no effect.
- **Localized Name** – The name that appears in the Blueprints and Area Contents lists.
- **Resource Name** – Is a unique, 32-character name that serves as a blueprint identifier.
- **Tag** – The string used to reference the static camera from a script or conversation.
- **Template ResRef** – The resource name of the blueprint template that this object inherits from.

### Scripts

- **Variables** – This is used to define and initialize local variables for the static camera.

Note that you can hold down the Alt key right click, and apply a slight roll to the camera for a slanted view. However, this feature doesn't seem to work quite right because the heading changes to a fixed value. Still it is interesting to experiment with this and the other properties in the Misc section.

## Lights

---

The Light blueprints can be used to place point light sources in areas. These are useful for simulating lit street lights, the glow from a fireplace, burning torches along gloomy cave tunnels, the illumination emitted by a window, forge or lava, or for producing a mysterious magical glow coming from an enchanted site. Lights can also be incorporated into placed effects to produce, say, a glowing candle. Unfortunately there are no standard light types included in the toolset, so you will need to make your own set.

The positioning of lights can be used to highlight certain locations. Lights should almost always be positioned so that they appear to originate from a source, such as a placed effect or a lamp post, and the radius and intensity of the illumination should be in balance with the strength of the source. Thus the light of a candle should be smaller and less luminous than the light from a bonfire.

For example, a flickering light source can be positioned next to a 'Fireplace' [placeable](#). For further realism, a burning 'Fireplace' [placed effect](#) and a 'Fireplace' [sound](#) can be positioned on the logs. The various 'Lit' prefabs can provide dynamic light sources, or you can use the windows along the walls of [interior areas](#).

The color of the light can elicit certain associations and thus should be chosen with some care. This is particularly true of red lights, which tends to draw the eye even when they are only a small source. Red, orange and yellow colors can appear warm and energetic, while green, blue and violet are cool and placid. Combining warm and cold lights can create nice contrasts and make an area more visually interesting. However, it is a good idea to use only two or three colors in an area; any more and the effect can appear haphazard.

For interior areas, point lights source heighten the surface texture on walls and floor, causing the details to stand out. Unfortunately, the light from a point source is not interrupted by interior walls. If you position a light such that the range reaches beyond a wall, then the far side will be illuminated as well. Needless to say this can make a scene



## Lights

appear unrealistic. To address this you can either place lights in locations where the illumination will not cross a two-sided wall, or reduce the range of the light so that it illuminates a smaller area. To check for this, you can turn on the 'Light Spheres' option under the 'Show/Hide' filter menu in the toolbar. A third alternative is to use matching light sources on both sides of the wall.



*Note how the light source (top center) spills through the wall surface and does not cast shadows off the barrel or trunk.*

I have experienced inconsistent results with shadows being cast by creatures or placeables. Within the toolset, the floors of Standard Interior, Illefarn and Castle tiles show shadows being cast, while the Estate and Sunken Ruins tiles do not. (An exception is when a point source is placed at ground level on an Estate tile.) The rendering of shadows in the game will depend on the client's graphics settings, and the game engine can also spontaneously turn off shadow rendering. Hence, you shouldn't rely on shadows always being available.

The intensity and color properties of a Light can be set to vary in a cyclical manner. This, for example, can be used to reproduce a wavering glow from a candle. The flicker variation makes the intensity of the illumination change over time as it cycles between bright and dim. Alternatively, the 'lerp' (short for *linear interpolation*) settings allows you to provide an alternate color and intensity, so that the light will vary smoothly between the Color and Lerp values. Note that only one of the lerp or flicker behaviors can operate on a single light source. If lerp is enabled then flicker will be disabled.

As noted in the toolset help guide, lighting can be

processor intensive. It is recommended that you have no more than three light sources on any tile or placeable at one time. The directional light cast by the Sun/Moon counts as a light source for this purpose. It may also help to lower the shadow intensity setting to a value of 0.6-0.8, thereby reducing the computation required.

### Properties

When a light is selected in the Selection panel or in an area, it's properties will be displayed as a list of parameter names and values in the Properties panel under the Properties tab. The properties are subdivided into Appearance, Basics, Behavior and Scripts blocks. When a light is in an area, a Misc block will be inserted that will include the location information. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

#### Appearance

- Casts shadow? – If true then the light should cast shadows. However, this may have no effect in interior areas as shadows are not cast from walls, and might not be cast from placeables.
- Color – This expandable field allows you to set the color and intensity of the light emitted by the object. The DiffuseColor setting is the color that is projected on the surroundings. The Intensity value is the brightness of the light within it's Range. The default Intensity is 1; a value of 0 turns off the light and values above 2 produce saturation. A value of 0.8 appears equivalent to the lit torch item.
- Lerp Target Color – This expandable field allows you to set an alternate color and intensity range of the light emitted by the object. The fields are identical to the Color settings above. These colors are only used if 'Lerp?' is set to true.
- On? – The object will only emit light when this is set to true. You can toggle this state in a script with the SetLightActive function. Thus you could, for example, turn lights on at dusk, then turn them off again at dawn.
- Range – This is the maximum distance that the light's

## Lights

diffuse color will reach. The light intensity will steadily decline with distance from the center until the point where it reaches the maximum range. You can visually display this radius as a sphere of red dots by selecting 'Light Spheres' from the 'Show/Hide' menu in the toolbar.

- **Shadow Intensity** – This is a decimal value between 0 and 1 that is used to determine the intensity of shadows cast by the light. Lower values, close to zero, will make the game run more smoothly.

### Basics

After a light is placed in an area, several of these fields are moved to the Misc block.

- **Classification** – This determines where the light will appear in the blueprint tree. Use the pipe character '|' to add sub-nodes.
- **Localized Description** – The light can not be selected by the player, so this is not used.
- **Localized Name** – This is used as an identifier in the blueprints list.
- **Resource Name** – Is a unique, 32-character name that serves as a blueprint identifier.
- **Tag** – The string used to reference the light from a script or conversation.<sup>50</sup>
- **Template ResRef** – The resource name of the blueprint template that this object inherits from.

### Behavior

This block controls the variation of the light source over time. You can use flicker or lerp, but not both.

- **Flicker Rate** – Flicker occurs in cycles, with the repetitions per seconds determined by this decimal value. Thus a value of 2 will cause the flicker to occur twice per second, while a value of 0.5 causes a flicker every half second.
- **Flicker Type** – This option determines how the light energy will vary during a cycle. Bounce spends more time near peak intensity, Jumpy results in sharp light

changes and Linear gives a smoothly varying pattern.

- **Flicker Variance** – The flickering can add this amount to the intensity at peak variance. A value of 0 results in no flickering.
- **Lerp Period** – This is the time in seconds for the light to cycle between the Color and Lerp Target Color.
- **Lerp?** – The light colors will vary in a cyclical manner by changing smoothly between the Color and Lerp Target Color. The transition is made using a blend of the two colors.

### Scripts

- **Variables** – This is used to define and initialize local variables for the light.

## Sample Light Sources

The following values for Range correspond to the radius of the light sources listed in the version 3.5 System Reference Document:

Light Source	Range
Candle	2
Lamp	10
Torch	12

The table below can be used to set the diffuse color of a light source so that it will approximate the appearance of a natural or artificial light source. Selecting the DiffuseColor field of the light placeable will bring up a color edit form. A suitable hexadecimal value can be entered into the HEX field of that interface.

Light Source	Hex Code
Candle	FF9329
40 Watt bulb	FFC58F
100 Watt bulb	FFD6AA
Sunshine	FFFFFFB
Cloudy day	C9E2FF

<sup>50</sup> There is a known problem where the 'Tag' strings for Light placeables are not restored from game saves. For details, see the notes for the 'SetLightActive' call in the second volume.

## Lights

### Windows and Curtains

Some interior tile variations include one or more window bays along a side. You can use the light blueprints to create a daylight glow coming from these windows.

#### Example

In a standard interior room with a window looking out over an area with heavy, wind-swept clouds, a Light blueprint is created that includes the following properties:

- Color – 125, 167, 217
- Intensity – 0.6
- Range – 5
- Shadow Intensity – 0.8
- Flicker Rate – 0.1
- Flicker Type – Bounce
- Flicker Variance – 4
- Flicker? – True
- Position No Snap – ... , 2.5

The light placeable should be located so that it is aligned with the middle of the window and touching the surface. This positioning eliminates most of the reflection from the wall's window texture and creates a reflected glow along the bottom of the window frames.

For curtains, a fainter light is positioned near the placeable so that it illuminates most of the surface, causing the curtain to glow faintly. For properties, in the same environment as the above example, the following are included:

- Color – 167, 167, 167
- Intensity – 0.4
- Range – 2
- Shadow Intensity – 0
- Flicker Rate – 0.1
- Flicker Type – Bounce
- Flicker Variance – 1
- Flicker? – True

Of course, you will want to create light sources that are specific to your own setting and environment. In an area of cloudless sky, for example, you will likely not want to use the flicker properties at all.

### Trees

---

The Tree blueprints can be used to create foliage in both exterior and interior areas. There is a large variety of different tree and bush species provided, with many duplicating a known type of flora. (Blueprint names with a capital 'C' at the end form a cluster of trees.) The shape and height of a tree object is randomly generated using the value of the Random Seed field. However, there is no way to know ahead of time what form a tree will take when a random value is entered; you will just have to experiment until you get a shape you like. Another limitation of this object type is that it can not be rotated.

Within the game, trees and shrubs can not be selected by the player and they behave as an environmental object that creatures can walk through. To turn trees into obstacles, you can use the [walkmesh cutter](#) from the Triggers blueprints. Alternatively, on outdoor maps, you can make the tree's grid mesh non-walkable.

Some of the [plugins](#) provided by the gaming community (such as PowerBar) can be used to give the trees in an area a random seed. This will save you the work of having to enter a different random seed for each tree and shrub. Plugins can also be used to automatically generate a [walkmesh cutter](#) around the base of a single tree.

By default, a tree is configured to fade from sight as a player character draws near, turning the object into a ghostly image. This has the benefit of preventing the tree from obstructing the sight of the player. However, it can appear somewhat unrealistic because the location of the core trunk is also obscured. A work-around is to place one of the 'Tree Trunk' placeables from the NATURE PROPS at the location of the tree's base.<sup>51</sup> An alternative is to use the 'Tree, Stump (03)' placeable.

To enhance the look of a placed tree, try modifying the texture underneath to simulate the impact the tree has on the

---

<sup>51</sup> For best results, you will need to change the scale of the placeable so that it will fit inside the tree, then lower it a notch or two and convert the trunk into an Environmental Object. For example, with a Maple tree, I scaled down a 'Tree Trunk {S- GreenAsh (X1)}' placeable to 55% of normal.

## Trees

environment. Hence, for a large tree in a grassy region, some dirt-textured underneath can reproduce the effect of regular shade from the leaves. Other possibilities include creating a small mound in the terrain underneath an tree, or placing some grass and/or rocks about the base about a solitary tree. For a distant tree skyline, the Rural Trees cards in the NATURE PROPS section of the Placeables can be added as a background, although you may still want to place a few scattered trees in the foreground to add depth.

### Properties

When a tree is selected in the Selection panel or in an area, it's properties will be displayed as a list of parameter names and values in the Properties panel under the Properties tab. The properties are subdivided into Appearance and Basics blocks. When an encounter trigger region is in an area, a Misc block will be inserted that will include the location information. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

#### Appearance

- Appearance – This is a menu of tree appearances. Some of the trees and bushes are created in stands.
- Casts shadow? – Define the types of shadows cast by the tree.
- Fade – If this is set to true, the image of the tree will fade when it would otherwise obstruct the line of sight to the PC. Setting this to false will prevent fading.
- Random Seed – This number is used to generate the tree. Changing the value will cause a new tree form to be generated. Thus you could place a row of trees then modify the seed for each object, resulting in a unique appearance for every tree. (It is usually sufficient to modify one of the digits or add an extra digit at the end until you get a look you like.)
- Receives Shadows – This determines the light sources from which the tree receives shadows. Note that rendering tree shadows is processor intensive, so you may need to turn them off if you have an area with a lot of trees. Trees outside the walkable mesh can usually have their shadows turned off.

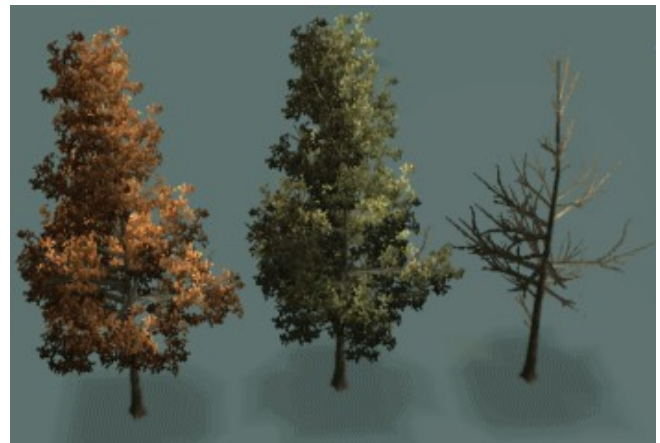
- Scale – This can be used to scale a tree's dimensions. This can be useful, for example, when creating potted plants by scaling down palm trees.

#### Basics

After a tree is placed in an area, several of these fields are moved to the Misc block.

- Classification – This determines where the tree will appear in the blueprint tree.
- Comment – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing.
- Localized Description – A tree can not be selected by the player, so this is unused.
- Localized Name – The tree can not be selected by the player, so this is only used in the blueprints list.
- Resource Name – Is a unique, 32-character name that serves as a blueprint identifier.
- Tag – This is not used because a tree is a static object in the game and can not be moved or modified by a script.
- Template ResRef – The resource name of the blueprint template that this object inherits from.

### Seasonal trees



*From left to right, the fall, summer and winter variants of the Columnar Oak tree using the same Random Seed.*

*The tree branches are identical in all three copies.*

Some of the trees have corresponding blueprints in more than one season, although none of the trees are present in all

## Trees

four seasons. In many cases, these tree variants will produce the same general tree structure when the same Random Seed is used, although they may differ in scale. Hence, this feature could allow copies of a wooded area to be modified to represent different seasons.

Here are the trees that have variants in more than one seasonal category:

- *Spring and summer trees* – Big Thorn, Bottle Tree, Crepe Myrtle, Japanese Maple and Pistachio Tree.
- *Spring, summer and winter trees* – Cherry Tree, Live Oak.
- *Spring and fall trees* -- American Elm, Aspen, Aspen Cluster and Elm.
- *Summer and fall trees* – Angelica, Bigleaf, Black Gum, Bradford Pear, Columnar Oak cluster, English Oak, Ficus, Horse Chestnut, Kousa Dogwood, Linden, Red Oak, River Birch, Sourwood, Sycamoor, White Birch and Willow.
- *Summer and winter trees* – Baobao, Bluegum, Buckeye, Honey Locust, Maple, Pin Oak.
- *Summer, fall and winter trees* – Columnar Oak, Green Ash, Grey Birch, London Plane and Sugar Maple.
- *Evergreen and snow* – Douglas Fir, Douglas Fir cluster, East Red Cedar, Fraser Fir, Fraser Fir cluster, Longleaf Pine, Longleaf Pine cluster, Monterey Cypress, West Red Cedar, White Pine, White Pine cluster.

To simulate all four seasons, the trees from the *summer, fall and winter* list could be used, with the summer trees also being used for the spring season. Likewise, trees from the *evergreen and snow* list can be useful for sub-arctic or mountainous regions. The Unique category has a few burnt editions of trees, so the following could be used to simulate the before and after appearance of a forest fire:

- Green Ash – summer, fall, winter and unique.
- West Red Cedar – evergreen and unique.
- White Pine – evergreen and unique.

## Placed Effects

---

Placed effects are objects that can be added to an area to produce a persistent, dynamic visual effect, such as a burning flame or falling leaves. The game comes with a set of stock placed effects that will function properly within the toolset as well as in the game. Within the editor, a placed effect will appear with a yellow helper cube that allows it to be selected and located. This cube can be selected just like any other placeable. When a placed effect is copied to an area, the effect will immediately begin to animate within the toolset. If you find this distracting, the effect displays can be turned off by disabling 'Placed Effects' in the Show/Hide menu of the toolbar.

Placed effects that are created using the 'Create Blueprint' menu item in the Selection panel may not always work properly. Instead, the desired result may be achieved by making a copy of an existing placed effect blueprint and modifying the properties to use the new special effect. However, the results are not consistent; failing, for example, with `fx_snow_fog`. See the Effects Files section of the second volume for more information.

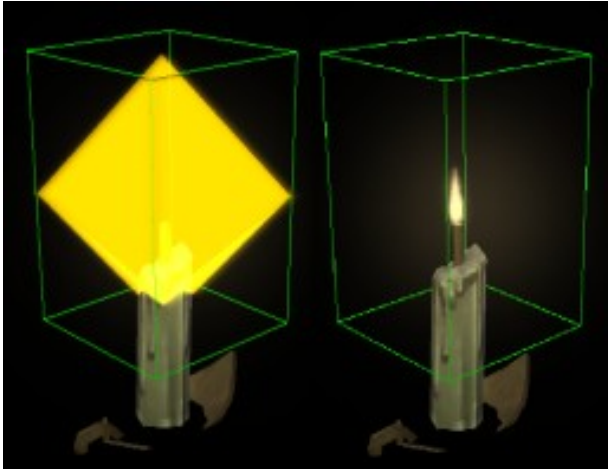
Creating a realistic light source can involve joining multiple blueprints into a single point. For example, the 'Lit Standing Torch' from the Prefabs provides a gas flame. The illumination from this light source must be supplied with a 'Lights' object, which can be placed at the height of the flame. This light may then need to be modified to reduce the shadow on the ground and given some flicker variation. Finally, placing a sound object taken from the "Environmental, Fire" group called 'Torch Fire Small' at the height of the flame will produce a burning sound near the torch.

A small Placed Effect may prove difficult to place properly because the effect is contained within its yellow helper cube. A particular example of this is the Candle placed effect. Suppose you want to align the flame of the Candle placed effect with the wick of the Candle placeable. With the Candle placed effect moved into the approximate position you want, select the effect to display its properties and then disable the 'Placed Effect Helpers' option under the



## Placed Effects

Show/Hide filter in the Toolbar. This will hide the helper cube surrounding the Candle's flame effect, making the flame position clearly visible. Now you can modify the values in the 'Position No Snap' Property to fine tune the location. Once you are satisfied with the placement, you can restore the 'Placed Effects Helpers' option then Group the effect with the placeable to keep them together.



*A Candle placed effect with the 'Placed Effects Helpers' filter enabled (left) and disabled (right)*

To create a new Placed Effect blueprint, select the Blueprints tab in the Selection panel, then open up the 'Empty' node. Next, pick an existing placed effect, such as 'Altar Glow', then right click and choose Copy Blueprint. This places a copy of the blueprint in bold font at the bottom of the node tree.

### Properties

When a placed effect is selected in the Selection panel or in an area, its properties will be displayed as a list of parameter names and values in the Properties panel under the Properties tab. The properties are subdivided into Basics and Behavior blocks. When an encounter trigger region is in an area, a Misc block will be inserted that will include the location information. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

#### Basics

After a placed effect is added to an area, several of these

fields are moved to the Misc block.

- **Classification** – This determines where the placed effect will appear in the blueprint tree.
- **Comment** – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing.
- **Localized Description** – Since placed effects can not be selected by the player, this field is not used.
- **Localized Name** – The descriptive name as it appears in the Blueprints list.
- **Resource Name** – Is a unique, 32-character name that serves as a blueprint identifier.
- **Tag** – The string used to reference a placed effect object by a script.
- **Template ResRef** – The resource name of the blueprint template that this object inherits from.

#### Behavior

- **Special Effect** – This field presents a pop-up menu with a long list of effect names. These are described in the Effects Files section of the second volume. For best results, select a continuous effect that is not intended for application to an object.

### Portals

The following effects can be used to creating a vertical magic portal with a placed effect:

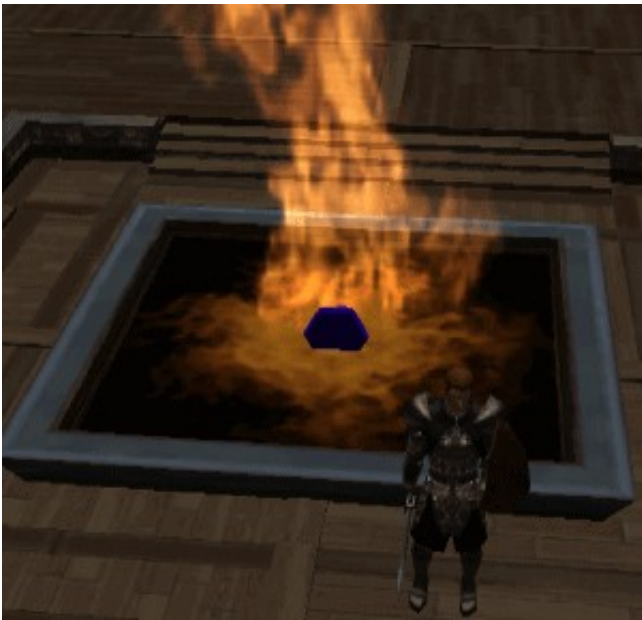
- `fx_b_fiery_song_portal` – 2-3' wide oval
- `fx_betrayers_gate` – 2-3' wide opening
- `fx_demon_portal` – 10' diameter circle
- `fx_kos_portal_*` – 6/3/1' diameter circle
- `fx_portal_gen_small` – 5' diameter circle
- `fx_portal_gen1` – 10' diameter circle
- `fx_shadow_portal*` – 3' wide oval
- `fx_song_portal*` – 3' wide oval

Most of these have a fixed orientation on the horizontal plane, so you will want to experiment with them before deciding where to place the effect.

## Placed Effects

### Example

In the example below I improvised a large vat of burning oil, as shown in the illustration. I didn't make it smaller because the Effect requires a certain size; it doesn't scale. This construct could be further enhanced by adding a black-colored water layer (with a high smoothness level) between the tile block and the placed effect, giving the oil a glistening look. A similar process can be used to create a natural burning oil pit.



*A vat of burning oil*

Environmental node has list of a Fire sounds, and from there I select *Fire Smoulder*. Most of the default settings are acceptable, so I fine tuned the maximum distance to 20 and the minimum to 3 by moving the view about and checking the sound levels.

7. Finally, I selected all four objects, right clicked and joined them into a group. The assembly is lowered to the desired elevation (to cover the sandbox's feet).

1. A Placed Effect is created using the *fx\_ashfire\_2* effect.
2. A *Sandbox* object from the Manmade Props list is scaled by [7, 7, 2] in the Properties list.
3. For the pitch black interior of the sandbox, I used the *{TileBlock2}* placeable from the Manmade Props list and scaled it by [0.5, 0.5, 0.13].
4. The tile block is centered within the sandbox by setting both to the same 'Position No Snap' values.
5. Add the placed effect and set it's vertical (z) position to 0.47 above the base of the sandbox. Lock the effect's height and carefully center the flame's base within the sandbox.
6. For a burning noise, I add a Sounds blueprint. The

## Prefabs

### Prefabs

---

Prefab blueprints are groups of placeables that can be added to an area as a prefabricated unit. These are useful for bundling together various placeables so that they can be repeatedly used, such as the furnishings for a bedroom. Once a prefab is copied to an area, it can be ungrouped by selecting the object in an area, right-clicking and choosing 'Ungroup'.

If you group several objects together and select them, you can right-click and choose Export Group to save the collection to a Prefab file. When you save the Prefab to the override folder, located in your documents under the game directory, the blueprint will appear in the list the next time you start the toolset. However, you will not be able to modify the properties of the resulting blueprint.

### Properties

When a prefab is selected in the Selection panel or in an area, its properties will be displayed as a list of parameter names and values in the Properties panel under the Properties tab. The properties are subdivided into Basics, General and Misc blocks. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

#### Basics

After a prefab is placed in an area, several of these fields are moved to the Misc block.

- Classification – This determines where the prefab will appear in the blueprint tree.
- Comment – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing.
- Localized Description – As a prefab is not selectable in the game, this field is not used.
- Localized Name – This is used as an identifier in the blueprints list.
- Tag – The string used to reference the light from a script or conversation.
- Template ResRef – The resource name of the blueprint

template that this object inherits from.

#### General

- Resource Name – Is a unique, 32-character name that serves as a blueprint identifier.

#### Misc

- Group – *Unknown*.

# Conversations

A conversation consists of a branching tree of dialogue strings that are used to perform social interactions with creatures or objects in the game. They can be used to yield information to the players, to provide personalities to the various inhabitants, initiate trade and to control the plot. Within the game, a conversation is typically displayed using a cut-scene interface that shows the alternating speakers at varying camera angles.

Text	Actions	Conditions
Root		
Yoo?	gc_clear_actions()	gc_global
Captain? Can I ask you a few questions?		
Certainly <FirstName>. I have a few moments to spare. What concerns you?		
What happened during the storm?		
Wait a stick up us all of a sudden like, in the middle of the second watch. The helm reported all's well, and		
[CONTINUE]		
It was a wretched first night. The crew had to scramble to draw in the sheets and clear the deck. The		
[CONTINUE]		
I tried to sail into the wind, but the cursed weather couldn't make up its mind and we took a few c.		
[CONTINUE]		
The crew began seeing odd things in the waves. A storm can play mean tricks on your eyes.		
That's strange indeed.		
At that point, it was like the storm was attacking us. While the waves swept over the de		
[CONTINUE]		
I lasted another two nights, but it was never as bad as that first. We were lucky ind.		
What happened to the crew?		
How is the ship?		
When will we be getting underway?		
Did you know that there is a stowaway in the hold?		
That's all I wanted to know	gc_global_int("w_	gc_global
What happened to the crew?		
Yes, well Patrick took a nasty fall to the deck and was knocked out cold for a day. Ivan got swept over the		
[CONTINUE]		
Peter's foot got hit by a falling crate, but he'll recover. Shanty was up clearing the rigging when the lg		
[CONTINUE]		
The worst was Hobbs. Three of the creatures that came over the side grabbed him and dragged		
That's awful. But where are the rest of the crew?		
Oh I sent two teams ashore to the island. Much of our water was fouled during the storm, so		
Shouldn't they be back by now?		
Aye, they are a tad late, for certain. Still, there's foul weather afoot and they may have t		
Why not send out somebody to look for them?		
Have you taken a look around. <lad/less>? We were short handed even before the		
Perhaps I could go?		
Absolutely not. You don't look like you could handle a small boat in this weath.		
What happened during the storm?		
How is the ship?		
When will we be getting underway?		
Did you know that there is a stowaway in the hold?		

*A sample conversation*

A "bark string" is a one-liner conversation that will be displayed as a floating string above the speaker. An extended dialogue consists of statements by the conversation owner, followed by a list of one or more possible responses by the player. The player's choices will serve to steer the conversation and to determine the consequences within the game. The various dialogue branches may result in a change to creature's attitudes toward the PC, exchanges of goods or information, the addition or completion of quests, gaining or losing a PC recruit, opening a store, revealing new locations on a map, transporting the party to a different location, or even the initiation of hostilities.

In the game, conversations with creatures are begun by clicking on the creature when a quote cursor appears, or by shift-right clicking and selecting "Talk To". For a placeable, you need to make it non-Static and Usable, then put a script such as "gp\_talk\_object" on the "On Left Click" property.

A conversation can be created in the toolset by selecting the Conversation tab of the A/C/S panel, then right-clicking and choosing 'Add'. Alternatively, a Conversation can be added from the 'New' sub-menu under the File menu. When a conversation is opened, an editor interface appears in the Edit panel. A toolbar is displayed at the top of the panel, a data table underneath, and an input and configuration section at the bottom.

The initial row in the conversation data table is always titled 'Root'. A data table can contain one or more conversation trees, with each beginning at a node underneath the root. Each branch in a conversation tree is right indented from it's parent. Within the game, a conversation proceeds down the branches of a tree until it reaches a conclusion at an [END DIALOG] entry.

The red rows in the data table represent statements by the owner of the Conversation or by NPCs, while the blue text is used for replies by the player. An empty [CONTINUE] line results in a drop through to the next line by the current speaker, allowing an extended monologue.

Text	Conditions
Root	
Ah, you're finally awake are you, young <man/woman>...	gc_global_int("w_mr_talk_captain", "=0")
Generic conversation[You are looking better now. Goo	gc_global_int("w_mr_talk_captain", "=0")
[Ask first time]Hello again, <FirstName>	gc_global_int("w_mr_talk_captain", "=1"). And
[Ask again]Are you finally ready to perform my task <Fi	gc_global_int("w_mr_talk_captain", "=1"). And
[Final time]<FullName>, you will apologize or leave my ...	gc_global_int("w_mr_talk_captain", "=1"). And
[END DIALOG] Get out of my sight!	gc_global_int("w_ask_powder", "=3")
Do you have my bag of Quinox?	gc_global_int("w_mr_get_powder", "=1"). And
Oh dear, is that fighting I heard? Quickly, <boy/girl>, he...	gc_global_int("w_mr_pirate_attack", ">0"). And
What happened out there? I was worried sick!	gc_global_int("w_mr_pirate_attack", ">0"). And
Hello again <FirstName>	gc_check_item("w_book_riddle_south", 0)
[END DIALOG] Remember <FirstName> stay out of tr...	

*A conversation data table with multiple separate discussions*

The conversation data table can contain multiple separate discussions with the same owner. The choice of which discussion to use is controlled by the configuration properties at the bottom of the Edit panel. The 'Show Once?' field under the Node tab will determine how often a discussion will appear during the game. Thus, setting this to 'Once per game' will only allow that node to be used once.

The Conditions tab can be used to apply more complex logic for displaying a discussion node, such as checking the value of a variable that is being used to track the state of a quest. (See the [Conditions](#) section below for more details.) Whenever a discussion is initiated with a creature, the game

## Conversations

will display the first conversation node under the Root line that currently satisfies the 'Show Once?' setting and the Conditions.

### Properties

When a conversation is selected in the A/C/S panel, it's properties will be displayed as a list of parameter names and values in the Properties panel under the Properties tab. The properties are subdivided into Behavior, Comments, Scripts and Voiceover blocks. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

#### Behavior

- Default TGA – *No effect.*
- Delay per conversation entry – *No effect.*
- Delay per conversation reply – *No effect.*
- Multiplayer Cutscene – This is set to true for conversations with more than two participants. When true, the players will be unable to control their characters while the conversation is in progress. See the [Node](#) section below for information on how to set the current speaker.
- Neverwinter Nights 1-style – The first version of NWN used a dialog interface for conversations, rather than the cut-scene panels. Setting this to true will force the use of a dialog for the interaction. This may be useful, for example, when conversing with an object.
- OverrideXML – in SoZ, for party chats this is set to "partychat\_ol.xml".
- PartyChat – this enables the party chat mode used in the SoZ campaign.
- Prevent Zoom – This is marked as obsolete.

#### Comments

- Comments – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing.

#### Scripts

- Abort Conversation Script – This script will be run if the conversation is terminated before reaching an

[END DIALOG] entry. A termination can occur when the Neverwinter Nights 1-style is being used and the PC moves out of range of the conversation owner.

- End Conversation Script – This script is run upon reaching an [END DIALOG] entry.

#### Voiceover

The following are used for lip syncing during voice-overs.

- Default Speaker Appearance –
- Default Speaker Gender –
- Default Speaker head Variation –
- Use Default Speaker Appearance For Lip-Sync Animations –
- Voiceover Character Name –



## Editing

### Editing

A new entry can be added to the conversation data table by selecting an existing entry in the data table, then choosing 'Add' from the conversation toolbar. Likewise, an existing entry (and all of its sub-entries) can be deleted by selecting that entry and choosing 'Remove' from the conversation toolbar.

When the conversation data tree starts to grow, it can be convenient to contract selected nodes by clicking on the small '-' icon at the start of the node, or expand a node by clicking the '+' icon. There are also options in the toolbar for controlling the display of multiple nodes:

- Expand All – This will expand any collapsed nodes in the conversation.
- Collapse All – This will collapse all nodes in the conversation into the root node.
- Expand From Here – This will expand all nodes from the selected line downward.
- Collapse From Here – This will collapse all nodes from the selected line downward.

Underneath each red entry, you can add more than one reply. Within the conversation interface of the game, these entries will be displayed as a list of potential replies, and the line selected by the player will determine the branch of the conversation to follow. The order of the entries under a node can be rearranged by selecting a branch and clicking on 'Move Up' or 'Move Down' in the Edit panel toolbar.

You can apply various criteria to each row that will determine whether it will be displayed in the game. (See the [Conditions](#) section below). When there is more than one red entry under a blue row (or under the root node) then the conversation will follow the first branch that satisfies the conditions for it to be displayed. Thus you can have multiple red rows, each with a different condition, and the game engine will check each row in turn until the conditions for a row are satisfied. This can be used, for example, to follow separate conversation branches depending on the outcome of a skill check by the player. It can also be used to manage the conversation based on the state of the plot, the reputation of the player, the particular

biases of the conversation owner, and so forth.

### Links

Linking is a method of replicating a conversation branch to multiple locations within a data tree. To link to a conversation branch, first select that branch and left-click 'Set Link Destination' from the Edit toolbar. Next, select the entry under which you want the link to appear. (If the destination was red then you must select a blue line, and vice versa.) Clicking 'Insert Link' will add a gray text row that is linked back to the destination. Any edits made to the destination entry will also be applied to the linked rows.

Links can be used to efficiently manage a list of player questions. After the player selects one of the questions from the menu, the conversation proceeds down the branches until the question is answered (or not). At that node, the remaining questions can be linked under the conversation owner's response. This can be repeated for each of the questions, and the conversation will cycle through the queries that the player wants answered, then exit at an appropriate selection. ("I have to go now", for example.)

The main drawback to using links is that they can accidentally result in perpetual loops. It requires some planning and verification to prevent this situation. One method that can help is to ensure that no sub-branch of the linked node is linked back to a higher level in the conversation tree. That is, if a linked destination node is at the fourth indent on the tree, check that none of the node's sub-branches is linked at the first, second or third indent.

Text
[-] Root
[-] Can I help you?
[-]  Question #1 Where is the Inn?
[-]  At the top of the hill.
[-]  Question #2 Which way to the docks?
[-]  END DIALOG  Thanks, I have to go.
[-]  Question #2 Which way to the docks?
[-]  Follow the road down.
[-]  Question #1 Where is the Inn?
[-]  END DIALOG  Thanks, I have to go.
[-]  END DIALOG  Thanks, I have to go.

*A trivial example of a cross-linked Q&A conversation.*

*Note the comments between the pipe characters.*

The node tab contains a field called 'Show Once?' that can

## Editing

be used to determine how often a line will appear. This can be used, for example, to prevent already-answered questions from appearing again. Make sure that at least one of the responses will always appear, however, or the conversation may be aborted when none of the responses can be displayed.

When you try to delete a node in the data table that has one or more link destinations, a warning message will appear giving you an opportunity to abort the action. If you proceed, both the node tree and any links to rows in that node tree will be deleted. As it may be difficult to remember what links you had set in an extensive conversation table, it is usually a good idea to replace the links first before proceeding with the delete. One method of identifying these links (in situations where the same line of dialogue was used) is to add a temporary comment to the destination row. Double-clicking on a link will take you back to the original entry.

### Text Input

A line of text on the data table can be edited with the input box at the lower right of the Edit panel. If the input box is too small, you can resize it by moving the cursor to the borders of this input area, where the cursor will change to a horizontal or vertical bar with two arrows, then left-click and drag.

Along the top of the input area is a pull-down menu for the intended language; this can normally be left alone. Underneath is a button labelled 'Edit String Ref...' that is used for editing string references from the dialog.tlk file. I believe this file is used to store the strings that will be translated during language ports.

The main edit box is used to input the text string that will be displayed during the conversation. This is a simple word editor that should accept ASCII characters, or whatever your language version supports. There is no spell checking available, but you can also paste strings here from system's clipboard, allowing you to use a full-featured text editor.

Any text segment placed between a pair of pipes, parentheses or curly braces will not be displayed during a conversation, but will still appear in the data table. This is

convenient for keeping notes while editing an extensive conversation, and is useful for distinguishing identical lines and linked entries. You can hide any comments by disabling the 'Show Comments' field in the Edit panel toolbar.

At the bottom of the edit area is a pop-up menu next to an Insert button. This menu contains a number of character-specific tokens that the game will configure appropriately for the speaking character.<sup>52</sup> Thus, if you select 'him/her' and click Insert, the token '<him/her>' will be placed at the current edit point, and in a conversation either 'him' or 'her' will appear depending on the gender of the PC. (For example, you might want the conversation owner to say 'Guards, take him to the dungeon', if the PC is male, or use 'her' for a female PC.)

The input box can allow a long string of text to be entered for a conversation line. However, the area available for strings in a cut-scene is not unlimited. Typically it is a good idea keep the length of the speaker's entries to about 200 characters or less, and the PC responses to under 100 characters.

### Font Format

The conversation's font can be formatted by means of enclosing HTML-like tags of the form: `<format>some text</format>`, where *format* is one of the following:

- 'italics' or 'i' for *italicized* text.
- 'bold' or 'b' for **bold** text.

Color text can be created using a `<c=Color>some text</c>` tag pair, where *Color* is either a valid color name or a '#' followed by a 6-character hexadecimal value. The valid color names are listed in the 'nwn2\_colors.2da' file; the uppercase versions of these color names will also work.

The same Color Edit Form that is used to set tints can also be employed to find the hexadecimal value of a particular color. To use this method, select a color from the color palette then write down the value listed in the HEX output box. For example, the last color in the palette array along the right side of the form is called Darker Warm Brown (per the mouse-over pop-up) and has a hexadecimal value of

---

<sup>52</sup> There are additional custom tokens available. See the 'ga\_refresh\_timedate\_tokens' script, for example.

## Editing

'603913'. The following string will display brown text:

```
<c=#603913>Darker Warm Brown</c>
```

Note that these font formatting techniques can also be applied to item names and descriptions in their properties fields, or to the map note text in a waypoint.

## Tabs

---

At the bottom of the conversation Edit panel is a set of four tabs: Conditions, Actions, Node and Animations. These will change the configuration options that appear in the lower left part of the Edit panel. The Columns selection of the panel toolbar will display a menu that can be used to select which of the configuration values will be displayed in the data table. Thus, the scripts and script arguments selected under the Actions tab can be displayed in a column by selecting 'Actions' from the Columns menu.

## Conditions

When this tab is selected, a table with a green background will appear. This can be used to configure a sequence of condition scripts that will be run prior to displaying the selected line in the data table. Clicking on the 'Add' option above the green table will add a row to this table, while 'Remove' will delete the selected row. The Move Up and Move Down options can be used to rearrange the rows.

The toolset's built-in condition scripts use a 'gc\_' prefix. These are described in the Scripts section of the second volume. To select a condition script, choose a row and click on the down arrow under the Script column. You will need to click on the 'Refresh' button in order to make the script's input argument fields appear. If the 'Preview' option is set above the condition table, the condition script for a selected row will be displayed in the field below the condition table. (It may be helpful to expand the display by dragging on the borders.)

You can control the evaluation of the script by means of the buttons to the left of the Script column. Clicking on the 'And' button will toggle it to 'Or'. Setting the 'Not' button to true will cause the line to evaluate to true if the script returns false, and vice versa. The conversation entry will be displayed if the sequence of script logic evaluates to true.

For example, suppose you only want a conversation entry to appear if the PC speaker is a dwarf *and* has an Intimidate rank of 4 or greater. The first line of the condition table would use the 'gc\_check\_race' script with a sTarget value of

## Tabs

\$PC\_SPEAKER<sup>53</sup> and sRace value of “dwarf”. The second line would use the 'And' logic setting with the 'gc\_skill\_rank' script, an nSkill value of 11 (see the script comments) and an nRank value of 4.

The gc\_global\_\* and gc\_local\_\* scripts can be used to check the values of variables that were previously set in scripts. (The global variables are associated with the module, while local variables are assigned to a specific object such as the conversation owner.) These values can be used to track the state of the plot, and modify the conversation accordingly.

### Actions

The Actions tab will display a table with a pink background. This is used to configure any scripts that will be run when the conversation entry is selected by the player or spoken by the conversation owner. The scripts are run in the order listed within the table. To add an action script, select the 'Add' option. Selecting a row and clicking 'Remove' will cause the row to be deleted. The Move Up and Move Down options can be used to rearrange the rows.

The toolset's built-in action scripts use a 'ga\_' prefix. These are described in the Scripts section of the second volume. To select an action script, choose a row and click on the down arrow under the Script column. You will need to click on the 'Refresh' button in order to make the script's input argument fields appear. If the 'Preview' option is set above the action table, then the action script for a selected row will be displayed in the field below the condition table. (It may be helpful to expand the display by dragging on the borders.)

As an example, suppose you want the conversation owner to take an item from just the PC then give some gold in exchange. (A 'gc\_check\_item' condition script can be used to check if the PC actually had the item.) On the first entry, the script is set to 'ga\_take\_item', with the sItemTag field set to the tag of the item, nQuantity set to 1 and bAllPartyMembers set to 0. The second entry uses the

'ga\_give\_gold' script with nGP set to the amount of gold and bAllPartyMembers set to 0. These exchanges will appear in the player's chat window afterward.

The ga\_global\_\* and ga\_local\_\* scripts can be used to set variables. These allow the state of the plot to be tracked based on the variable values, and they can be checked by condition scripts during future conversations.

### Node

Choosing this tab allows a set of properties to be configured for the selected conversation row. The properties are subdivided into Behavior and Line blocks. Each block can be contracted or expanded using the small plus/minus box at the left of the header.

#### Behavior

- Animation – This field shows the animation setting selected under the Animation tab. See the [Animations](#) section below.
- Camera Settings – This expandable field can be used to configure the camera for the cut-scene display. See the Camera Shots section below.
- Delay – This is the minimum delay that must occur before the player can cause the conversation to proceed to the next line.
- Quest – This is used to update the player's journal.
- Show Once? – This field is used to control how many times the line will appear. The available options are Always, Once per conversation, Once per creature that uses this conversation, and Once per game.
- Sound – The prefix for one of the game's '.wav' files can be inserted here, such as for use as a voice-over. These can be found under the various Sound\* folders in the NWN2 program folder, or place a '.wav' file in your local document override folder.
- TGA to display – The prefix for a '.tga' file in the campaign or module folder can be inserted here. It will be displayed instead of the camera view.

#### Line

Most of these fields are used for voice overs and lip syncing. They can be ignored unless you plan to implement

---

<sup>53</sup> The target field recognizes key word strings that begin with a \$ symbol. These are defined in the 'ginc\_param\_const' include file. Thus, '\$PC\_SPEAKER' sets the target to the PC speaker.

## Tabs

that functionality in your module.

- **Comment** – This is a utility field for entering notes that will not be seen in the game, but which can be used during editing.
- **Link Comment** – *Unknown*.
- **Listener** – This can be set to the tag of the creature that the speaker is facing.
- **Needs VO** – This is a flag that is used for internal development by the toolset vendor.
- **Speaker** – The tag of the current speaker. If you want the conversation to include multiple participants, you can use this field to control which red line is spoken by each character. When this is empty, it defaults to the conversation owner or player, as appropriate.
- **Speaker Appearance** – This is set to the body type of the speaker.
- **Speaker Gender** – The speaker's sex.
- **Speaker Head Variation** – The head variation of the speaker, as set in the 'Appearance (head)' field of the creature's properties.
- **Text** – The text string as it will be presented during the conversation. This is basically a copy of the input box to the right of the node properties.
- **VO needs to be recorded** – This is a flag that is used for internal development by the toolset vendor.
- **VOPadding** – *Perhaps a time value?*

### Multiple Conversations

The 'Show Once?' field of the Node table can be used to control when a conversation branch will appear. If it is set to 'Once per game', for example, that line will only appear on the first occurrence. Thereafter it is treated as a non-existent branch. If a Conversation has multiple nodes under the Root row, and each one is set to 'Once per game', then every time a conversation is held with the owner, the next node under the Root row will be displayed.

A combination of the 'Once per game' field and the condition scripts can be used to fine tune what branches of the data table will appear during a particular conversation.

### Camera Shots

During a cut-scene conversation, the placement of the view is determined by the Camera Settings on the Node tab. For most conversations between the PC and an NPC, the default setting of Random works well. Occasionally, however, you may want to place the view so that it is facing a particular direction from a specific position. This can be implemented using the Static Cameras blueprint.

To create a static viewing position, select the Blueprints tab on the Selection panel and choose Static Cameras. Creating and selecting a Static Camera blueprint will place a green camera-like shape within the area. After a camera object is placed in the scene, select Preview from the Properties panel. This will display the scene as viewed through the camera lens. You can maneuver the camera about, change the height, then tilt and pan the camera object until the preferred view is obtained. (It's a good idea to set Position Lock to true once the location and height is set.) Finally, set the camera object tab to a unique name.

The next step is to place the camera scene into a conversation. Select the particular line in the conversation tree, then click on the Node tab. Open up the Camera Settings row and change Mode to Static Camera. When you choose the StaticCamera field, the popup menu will allow you to select the camera under the area name, based upon the camera tag. Finally, to make the scene remain in the cut-scene for a period of time, change the Delay field to however many seconds you want it to appear. The conversation will not advance until this time is complete.

If you want something to happen during the cut-scene view from the static camera, you can add scripts to the Actions tag. For example, the `ga_play_animation` script can be used to cause a creature to perform an animation.

### Animations

Each node in a conversation can have an animation associated with the speaker. First select the line of text and then the node tab. In the Animating Creature Tag list, select the creature to animate; usually the speaker. (A possible exception could be if you change the camera setting and want the viewed creature, as identified by its tag, to perform



## Tabs

an animation.) The listed animations are equivalent to those available with the `ANIMATION_FIREFORGET_*` constants via the `ActionPlayAnimation` function.

Next, select an animation from the Body Animation list. (Not every creature will be able to animate all animations, so you may need to experiment.) An asterisk will appear next to the selected creature indicating an animation has been selected. If you want to deselect the animation, select the creature and click the Clear button.

To view the current animation for a conversation node, click on the tag with the asterisk in the node interface. The Body Animation list will scroll to the selected animation.

The Facial Animation feature doesn't appear to work.

### Blurt Strings

Conversation entries that consist of a single line of speech are known as one-liners, or "blurt strings". These appear as a floating line of text over the creature. They can be used, for example, to provide a brief response to a selection of the creature by the player. Another use is as a random comment triggered by a heartbeat or on perception event.

Multiple one-liners can be put in a conversation file. To choose a line at random, you can use the `'gc_rand_1of'` condition script. The comments section at the top of the script explains how to use it. (To script a one-liner response, use the `SpeakOneLinerConversation()` function.)

### Intelligent Weapon Conversation

An [intelligent weapon](#) requires a Conversation in order to communicate with the PC. The name of this conversation should be the same as the weapon tag, and the conversation should include both one-liner messages and interactive conversations.

#### One-liner Messages

A one-liner message can be set to randomly appear when the intelligent weapon is equipped, unequipped or strikes an opponent. The odds for one of these message types to be generated is set by the percent 'CHANCE' configuration constants in the `'x2_inc_intweapon'` file. When a one-liner

message is triggered, the following two local integers will be set on the PC:

`X2_L_INTWEAPON_CONV_TYPE`

- equals 1 when item is equipped.
- equals 2 when item is unequipped.
- equals 3 or 4 when the item strikes a target.

`X2_L_INTWEAPON_CONV_NUMBER`

- when the type equals 1 or 2, this is set to a random integer from 1 to 5.
- when the type equals 3, this is set to a random integer from 1 to 20.
- when the type is 4, this is set to the result of a `GetRacialType` call that is run on the target.<sup>54</sup>

Within the Conversation, these local integers can be used by Condition rules to select the message to be presented. You can configure up to 58 one-liner exclamations: 5 for On Equip, 5 for On Unequip, 20 for On Hit, and 28 for On Hit against specific races. It is also possible to fire one-liner messages from a trigger by a call to `'TWPlayTriggerQuote'`. For an example of a triggered one-liner for an intelligent weapon, see the third volume of the NWN2 Toolset Notes.

To check for the specific message being called, the one-liner Conversation entry could perform a logical AND of two `'gc_local_int'` calls on the local integers listed above. For example, suppose there is a one-liner entry intended for an unequip event with conversation number 3. A condition check can be for a type 2 (unequip) conversation by passing the following parameters to `'gc_local_int'`:

- `sVariable = X2_L_INTWEAPON_CONV_TYPE`
- `sCheck = 2`
- `sTarget = $OWNER`

This can be logically AND'd with a second condition check of the unequip conversation number using a `'gc_local_int'` call with the following parameters:

- `sVariable = X2_L_INTWEAPON_CONV_NUMBER`
- `sCheck = 3`
- `sTarget = $OWNER`

---

<sup>54</sup> See the `'racialtypes.2da'` file for the valid return values.

## Tabs

The resulting Conditions entry should look like:

```
gc_local_int("X2_L_INTWEAPON_CONV_TYPE",
"2", "$OWNER"), AND gc_local_int(
"X2_L_INTWEAPON_CONV_NUMBER", "3", "$OWNER")
```

### Custom Condition Script

Note that it is somewhat inefficient to call 'gc\_local\_int', possibly twice each, for up to 58 consecutive Conversation entries. To improve the performance, I created a short Condition script called 'gc\_match\_iw\_msg' that works like the following:

```
// gc_match_iw_msg
int StartingConditional( int nMatch )
{
    int nConvType = GetLocalInt( OBJECT_SELF,
        "X2_L_INTWEAPON_CONV_TYPE" );
    int nConvNum = GetLocalInt( OBJECT_SELF,
        "X2_L_INTWEAPON_CONV_NUMBER" );
    return (nMatch == (1000*nConvType) + nConvNum);
}
```

This script multiplies the conversation type by 1000, then adds the conversation number and compares the total to an integer passed as 'nMatch'.

With this script in place, the valid ranges for nMatch are:

- 0: All interactive conversations
- 1001–1005: Equip one-liners
- 2001–2005: Unequip one-liners
- 3001–3020: On hit one-liners
- 4001–4030, 4250: Specific race hit one-liners
- 5001+: Trigger one-liners

The '4250' is for the Dwarf; based on a value of 250 set for the conversation number by the 'IWPlayRandomHitQuote' routine in 'x2\_inc\_intweapon'.

In the example earlier where the conversation type was 2 and the number was 3, the routine will only return TRUE when only when an nMatch value of 2003 is passed. The resulting Conditions entry in the Conversation tree will look like this:

```
gc_match_iw_msg(2003)
```

### Interactive Conversations

The Conversation file can include interactive discussions with the PC. These are triggered by the 'Talk to' selection in

the item pop-up menu. Prior to the start of a discussion, the local integer X2\_L\_IN\_INTWEAPON\_CONVERSATION will be set to TRUE on the PC speaker. When the conversation is complete, this integer will be set to FALSE. Hence, the start of every interactive discussion should include a condition check of type 'gc\_local\_int' with the following settings:

- sVariable = X2\_L\_IN\_INTWEAPON\_CONVERSATION
- sCheck = 1
- sTarget = \$PC

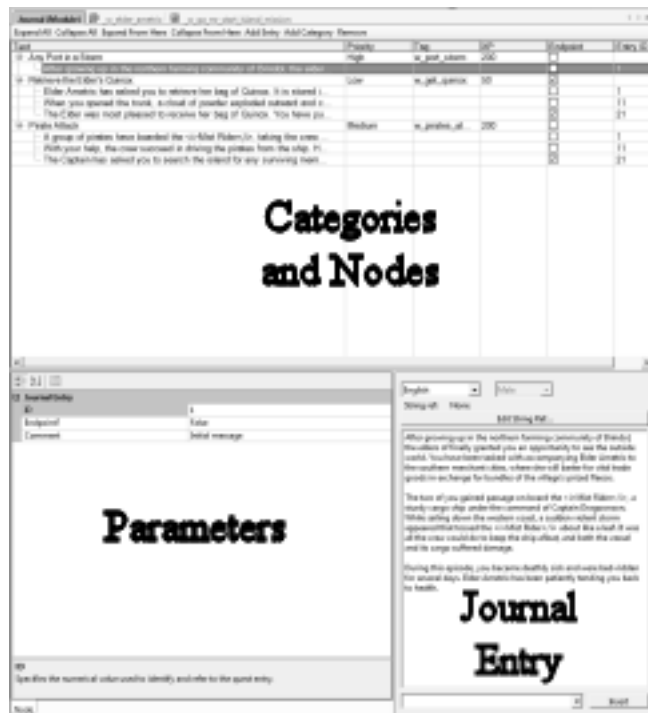
For best results, the interactive conversations should be moved to the top of the Conversation tree so that the game engine doesn't need to run all the condition checks for the one-liner conversations.

Each conversation between the PC and the weapon will use a new placeable to run the conversation. Hence, any state variables generated as a result of the conversation will need to be stored on the PC, rather than the placeable.

# Tools

## Journal

The journal is a log book for recording information related to player quests. In addition to providing colorful , expansive descriptions, it can serve to jog a players memory about the plot details. During a game the journal can be updated as specific events occur, such as an Action in a particular branch of a conversation or as the result of a command executed by a script.



*Journal interface*

The journal entries can be configured by selecting 'Journal' from the View menu. This places a copy of the Module journal editor in the Edit pane. The journal entry for a quest consists of a Category node with one or more entries underneath. The 'Add Category' option at the top of the editor can be used to create a new quest category. When a category is selected, the following input fields appear at the bottom of the editor:

- Comment – this can be used to record development

notes that will not appear during the game.

- Name – the string that will appear in upper case characters as the title in the game journal entry.
- Priority – a sort field that a player can use to organize their journal entries. It is selected by a menu and the possible values range from Lowest to Highest.
- Tag – a unique string that is used to identify the quest entry in conversations or script commands.
- XP – the experience point value awarded to the player character for completing the quest. Typically this is set relative to class level  $\times$  1,000 XP; the experience needed to reach the next level.

Each quest category can have one or more entries. These are the strings that are placed under the associated category. A entry is placed after the current selection using the Add Entry button along the top of the Edit pane. Each entry is assigned a unique identifier; typically in increments of 10 so that entries can be inserted in between.

When an Entry is selected, the following fields appear at the bottom left:

- Comment – this can be used to record development notes that will not appear during the game.
- Endpoint? – if this field is true, the quest will be flagged as completed when this entry is inserted into the Journal. At that point the XP will be allocated to the player character.
- ID – the identifier used to reference this entry in combination with the associated Category tag.

The message for an entry is edited in a text box to the lower right. This field can include paragraph breaks, blank lines and font formatting (see [font formatting](#) in the Conversation section). Every entry needs to be able to stand by itself in the journal because any previous entries are removed when the new entry is inserted. Thus a copy-paste may be needed to preserve some details between the entries.

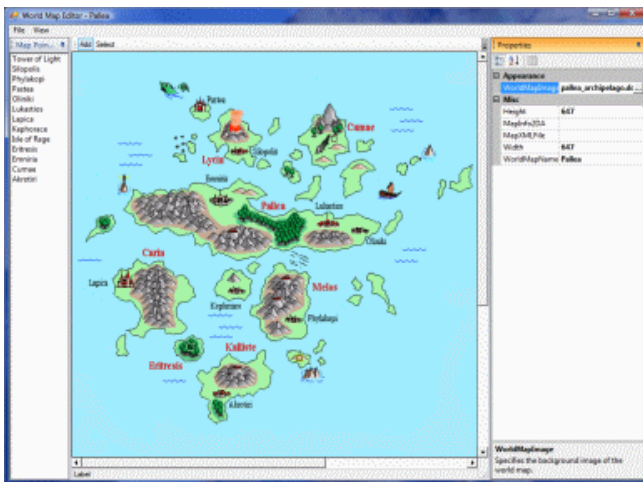
During a conversation, a journal entry can be added as an action using the `ga_journal` script, with the appropriate tag

in the *sCategoryTag* string and the ID in the *nEntryID* field. When the journal is updated, a message will appear in the middle of the conversation. The XP award for completing a quest can be granted with the *ga\_give\_quest\_xp* action script in the same conversation branch as the *ga\_journal* entry, with the appropriate tag in the *sQuestTag* field. (I like to include it near the end of the conversation so it is visible in the chat window afterward.)

In scripts, a journal entry can be inserted using the *AddJournalQuestEntry* command.

## World Map Editor

The World Map Editor plug-in (hereafter abbreviated WME) can be used to create a transition map for the player to move his party between different areas.



*A Campaign Cartographer image in the world map editor*

A world map is activated when the player clicks on a World Map Transition trigger, which is usually placed at egress points of an external area (such as a city gate). Clicking on this trigger will run the *'gtr\_world\_map\_cl'* script, which calls the *DoShowWorldMap()* routine from *'ginc\_worldmap'*. The trigger has a variable called *"sMap"* that is set to the name of the world map to display. This variable name should match a WMP file that was generated using the WME. (Hence, in the original campaign, the initial world map file is called *"Highcliff.WMP"* and the value of the *"sVar"* string variable in the World Map Transition trigger is *"Highcliff"*.)

The process of creating a World Map file and its associated scripts is explained in detail by Sunjammer's excellent *World Map Guide* v1.01 PDF document, available at the NWN Vault web site. The reader is referred to that document for guidance.

### Black Map Image

An issue when running the toolset in Windows Vista is that the WME map image may appear completely black. One solution for this is to load a new map in the following manner:

- Start the toolset in the default mode without opening a module.
- From the Plugins menu, select World Map Editor.
- From the File menu, choose Open... and open your module.
- In the World Map Editor, choose New from the File menu.
- Load a WorldMapImage file.

The image should now display properly. When you have done this once, the editor seems to run normally thereafter. However, if you open an existing WMP file, it may come up black again. In this case, just select the WorldMapImage file again and it should appear properly.

If the above approach doesn't work, there is an alternative. For this method, note that the black map image problem occurs when using a Targa image file, but it does not with a D3D/DDS file. However, the width and height of a D3D/DDS file is limited to powers of two, such as 512 or 1024. Hence, a work-around is to copy your  $647 \times 647$  Targa file image into the upper left corner of a  $1024 \times 1024$  D3D/DDS image, then do your editing using the latter file.<sup>55</sup> Note: make sure that all of your map points are located within the original  $647 \times 647$  image area and that the map width and height properties are set to 647.

<sup>55</sup> To avoid errors with the file format, I used a copy of the *world\_map.tga* file as the starting point. In PaintShopPro, I expanded the Targa file to  $1024 \times 1024$ , blanked it out, pasted in the new map and saved it to a new file name. Next I used the DDS Converter utility to transform it to a D3D/DDS format.

## World Map Editor

The D3D/DDS image file should appear properly when you first select it from the file open dialog by selecting the ellipsis button in the WorldMapImage property field. If you later re-open the world map, you may need to re-select the image file again for it to display properly. When you are ready to put the map into play, just change the WorldMapImage property to the original Targa file, then save.

### Overland Map

The overland map is a feature that was introduced with the Storm of Zehir (SoZ) campaign. It allows the player to be able to freely explore a large area of wilderness terrain,<sup>56</sup> which contrasts with the original world map approach where the player can only visit specific locales. The overland system uses exterior areas that have the OverlandMap option set in the area properties. While roaming the map, the player's camera position will remain fixed at the values set in the OverlandMap block of the area properties. (In SoZ, the camera is set to a distance of 25, a pitch of 40 and a yaw of 0.) The player's view is framed by a matte border and the objects on the map are scaled down to about 10-50% of their normal placeable dimensions. Likewise, the party is represented by a scaled down version of the current leader. To enhance the experience, there are reduced-scale placeables available from the OVERLAND MAP PROPS classification, such as buildings, towns, castles, forest canopies, etc.

To get started to creating an overland map, it is helpful to take a careful look at the overland areas in the SoZ modules. The traversable areas on the maps are overlaid in a jigsaw-like manner by trigger regions. Each of these regions runs a heartbeat script to handle the spawning of encounters, based on variables set on the trigger.<sup>57</sup> The trigger's On Enter script modifies the movement rate of the party for the terrain type and sets the background music.

The following local variables can be set on each terrain

trigger:

- nTerrain – this constant matches one of the nine terrain types defined in "ginc\_overland\_constants".
- nEncounterChance – the base percentage chance to spawn a random encounter. This is modified based on the game difficulty setting and the total number of active encounters is capped at 15.
- nEncounterTable – this gives a row number in the 'om\_encounter\_table.2da' file. The 'ENC\_2DA' column of this table lists the various overland map encounter 2da file prefixes. Hence, row 19 lists 'om\_enc\_g\_sam\_plains', which references the 2da file for the Samarlogh plains. The 'om\_enc\_\*.2da' files contain lists of encounters for up to five creature types per row.
- nRespawnTime –
- nTerrainBGM – this sets the background music for the terrain. The track is selected from the Music\_Track column of 'om\_terrain\_rate.2da' for the current terrain type.

Hidden locations and so-called "goodies" are generated at Ipoint placeables that run the 'gb\_hidden\_loc\_hb' heartbeat script. This script checks whether the PC spots the hidden location based on proximity and spot skill, then auto-generates the placeable object at the Ipoint location on a success. This placeable's tag is parsed from the Ipoint's tag, changing "\_ip\_to\_" into "\_plc\_to\_". For example, the Ipoint tag "g00\_ip\_to\_g23" becomes the placeable tag "g00\_plc\_to\_g23". In the SoZ campaign, the latter tag corresponds to a Cave placeable blueprint. When the player clicks on the placeable, it launches a conversation asking whether to enter the cave, which then runs the "ka\_olmap\_visit" script. This causes the party to jump to a waypoint that is parsed from the placeable tag (replacing "\_plc\_to\_" with "\_wp\_from\_"). The "g23\_wp\_from\_g00" waypoint is located in the "g23\_firenewt\_cave" area.

The neutral encounters that move along the roads during the game are generated by the overland map area heartbeat script. This uses the local variable "nEncounterTable" on the area to list a row in 'ENC\_2DA' for the neutral encounter 2da file prefix. In the case of the 'g00\_overland' map in SoZ, this uses the 'om\_enc\_g\_neutral.2da' file to generate the encounters. These are configured to follow the nearest [waypoint walk path](#).

The area heartbeat script 'nx2\_ol\_hb' looks for a local

<sup>56</sup> From what I can gather, the map scale is roughly 10 miles per grid square.

<sup>57</sup> See the nx2\_tr\_terrain\_hb heartbeat script from SoZ.



## Overland Map

string variable named 'sHeartbeatScript' on the area. This contains an area-specific heartbeat script that performs special encounter spawns. These are usually based on journal entries. See, for example, 'g00\_area\_hb'.

## Plugins

---

Various utilities are available from the NWN2 modding community that you can use to enhance the functionality your toolset. I'll describe a few from the NWVault web site that I have loaded and used.

When you download the plug-in, it may be in a compressed (zipped) folder or it could have a '.rar' suffix. In the latter case you will need to use an archive compression utility like [7zip](#) or 'Zip Genius' to extract the files. I used the 7z465-x64 windows installer to load the 64-bit version of 7zip and it worked properly in Vista.

Prior to loading *any* third-party plug-in, you need to run the Toolset, select 'Options...' from the 'View' menu, then set the 'AllowPlugins' setting to 'Load all plugins'. You will need to exit the toolset prior to updating any of the plug-in files in the NWN2 install directory.

There are other plug-ins available than those listed below, and I encourage you to explore what is available. The plug-ins can allow you to make edits in the toolset much more effectively, and some can provide enhanced functionality. However, you'll need to make certain that the plug-in will work with the current version of the toolset you are using. For example, some may work with MotB but not with SoZ.

### The Grinning Fool's Creature Creator

*Grinning Fool* authored this alternative creature creation wizard that you can use in place of the Appearance Wizard that comes with the toolset. The download is a zipped folder containing two '.dll' files and installation instructions. The following files must be placed in the game install folder:

- NWN2Toolset\Plugins\CreatureCreator.dll
- NWN2Toolset\Plugins\NWN2PluginToolsLibrary.dll

Once the plug-in is installed, you can select 'Launch Wizard' from under the 'Creature Creator' item in the Plugin menu. It will step you through the basics of creating a character, allowing you to choose the basic parameters, class and level, ability scores, saving throws, appearance, tinting and scale. However, you will still need to adjust the skills because it seems to assign more skill points than are

## Plugins

normally allowed.

I'm not sure whether this plugin is fully functional yet because the 'Next Steps' panel has several disabled buttons that I am unable to activate. You will still need to manage the creature's feats and skills through the Properties window.

### InCharacter

This plug-in by *Olblach* allows you import '.bic' files as Creature blueprints, or export blueprints as playable characters. A '.bic' file is generated each time you play the NWN2 game and create a new player character. Thus you can import characters that you have used in the game and turn them into an NPC.

The installation instructions come with a warning about making backup copies of your character files and modules before trying it. For further provisos, read the README file included with the download. The following file must be placed in the game install folder:

- NWN2Toolset\Plugins\InCharacter.dll

Once installed, an 'InCharacter' menu item will appear under the toolset's 'Plugins' menu. When selected, this will open a dialog interface. Clicking on Import will provide instructions on where to search for '.bic' files. The imported character will appear under the Creatures section of the Blueprints tab in the Selection panel. You'll still need to fill in some properties, including setting a tag, changing the faction ID and importing a script set.

### Lazjen's CPS Inventory Manager (CPSIM)

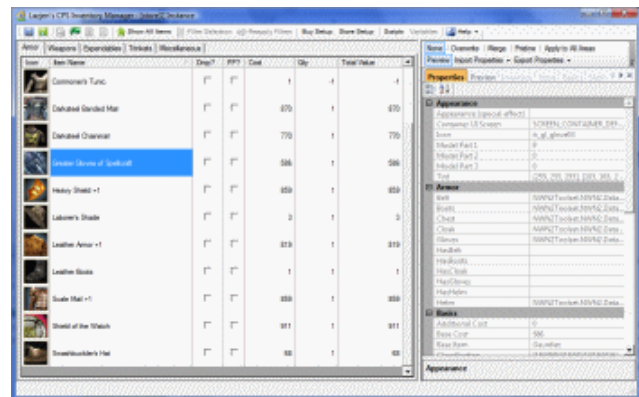
This plugin by *Lazjen* is delivered as a '.rar' file, so you will need an archive compression utility to extract the contents. The following file must be placed in the game install directory under the NWN2Toolset\Plugins\ folder:

- LazjensCPSInventoryManagerPlugin.dll

To use the plugin, you need to select either a creature, placeable or store (either an object or a blueprint) then choose the plugin from the Plugins menu. This will launch a dialog interface that allows you to manage the inventory in greater detail than is available with the toolset properties

window. Because of the large amount of graphics being loaded into memory, the interface can be somewhat slow to load and a little sluggish at first.

The main panel at the left of the dialog displays a tabbed, scrollable table of inventory, with the items listed in sort order by name. Upon selecting an item from the inventory or the list of all items, a panel on the right will display the properties of the item. The cost of each item is listed, along with the inventory icon. The only inconvenience is that it won't let you read lengthy property fields such as the Localized Description. For that you will need to use the toolkit blueprint view.



*Displaying a store's available armor inventory*

Along the top of the window is a toolbar of icons and options. These allow you to switch between inventory view or the full list, set various store parameters, and update the inventory item or blueprint. You can also filter the lists based on a cost range, regular expressions, and so forth. Unlike the toolset editor, this plug-in also arranges the available items into their proper categories.

### PowerBar

*Demium* wrote this plug-in that will perform a number of useful operations. The readme file in the download zipped archive lists the functionality provided. The following file must be placed in the game install folder:

- NWN2Toolset\Plugins\NetSpell.PowerBar.dll

After it is installed, an 'Open Module' dialog will appear when the toolset is launched, giving you an opportunity to select a module to open immediately. Modules that were

## Plugins

previously opened with this dialog will be listed in the table, making them easy to select. The plug-in will also load a new menu bar in the toolbar. Each item in the bar has menu of convenient functions for use in the toolset.

Here are some examples of the toolbar functionality:

- Object/Cut Walkmesh – this will create a [Walkmesh Cutter](#) around the base of a selected object. Note that this works for single [trees](#) but not clusters of trees.
- Object/Tweak – the scale, facing and tint of the selected objects is randomly modified within the selected range.
- Area/Randomize Trees – some or all of the trees in the area are given randomly generated seeds, allowing you to just copy the trees into place and then randomize their appearance afterward.
- Area/Rotate – this will rotate an interior area in 90° increments, then optionally bake the new area. Both tiles and objects are collectively rotated.
- 2da/Open – brings up a dialog that allows you to apply a filter rule to the 2da file list, then open a selected file.
- Browse/Icons – this is an icon viewer with the icons categorized into groups. You can also browse the icons you have assigned to item blueprints.
- Browse/VFX – preview the various '.vfx' files in a rendered pane.

## SpellPlug

This essential contribution by *Demiun* will check and fix the spelling and grammar of your text entries. You can use this plug-in to verify conversations, journal entries, object descriptions and the clipboard content.

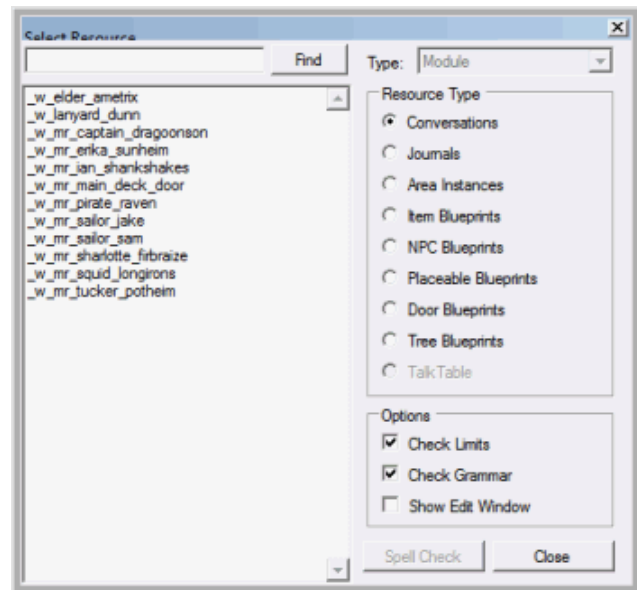
SpellPlug version 1.22 consists of the compressed folders SpellPlug.zip, SpellPlug\_dict.zip and SpellPlug\_gram.zip. The README in the first folder contains the installation instructions. The following files must be placed in the game install folder:

- NWN2Toolset\Plugins\SpellPlug.dll
- NWN2Toolset\Plugins\NetSpell.SpellChecker.dll

Once SpellPlug is activated, you will need to open the

'Options...' dialog under the View menu, select SpellCheck from the Plugins node, then set the 'Dictionary File' and 'Grammar Rule File' fields to the paths of the respective files.

To check the spelling and grammar of your game, click on the 'SpellCheck' option in the Toolbar and choose the 'Resources' menu item. This will open a modal dialog window that allows you to run checks on the various resources.



*Spelling checker resource selection*

As you perform a spelling check, you may find it easier to use the Enter key (rather than the mouse) to cycle through the checks and warnings. The main drawback with the interface is that, as long as no spelling issues get flagged, there is no means to cancel the spell check. For a lengthy conversation, you may have to cycle through every line.

Note that in Vista the text panels in the Warnings window displayed with black text on a black background. You can highlight the fields to read the messages. The warnings can be configured with the 'Options...' dialog.

## Incompatible Plugins

Each plugin release is compatible with a particular patch level of the toolset. This required patch level is normally listed on the download page. If you try to load a plugin with

## Plugins

a lower patch level than your current toolset, you will receive a 'Could not load plugin' error message and the plugin will not be available.

NWVault community member *Kethlak* has built a utility called 'Plugin Version Fixer' that can be used to make the toolset load the incompatible plugins. Be warned, however, that the plugin may not function properly with the patched toolset. (The other alternative is to load a copy of the toolset and patch it to the compatible level.)

Installing the Plugin Version Fixer requires modifying a small XML file and changing a pair of fields to match the current version of your Toolset. This is the version that appears when you open the Toolset, then select 'About...' from the Help menu. In my case the patched version number is '1.0.1588.0', but this will need to be changed when the toolset gets patched again. This file is copied to the game install folder:

- NWN2Toolset\NWN2ToolsetLauncher.exe

The following plugins required the above modification before they would load without an error.

- **Resizer** – this plugin from author *Bool* allows you to size multiple selected objects at the same time. You can increase or decrease the sizes by 10% multiples using the Home and End keys, respectively. (Thus, two consecutive Home keys will increase the scale by  $1.1 \times 1.1 = 1.21$ , or +21%.) The Backspace key will restore selected objects to their default scale. This plugin doesn't work on doors.
- **Tree Cutter** – author *sghctoma* wrote this plugin that will cut a set of selected Trees out of the walkmesh. It does this by generating a [Walkmesh Cutter](#) around the base of the trees. The tool also gives the trees a randomly generated seed. Unfortunately this plugin has not been updated since version 1.03, so it does not work with some of the tree types. (In one case it crashed my toolset.) Equivalent functionality is now provided by the PowerBar.

# Writing Scripts

The toolset has a built-in programming language that allows detailed customizing of a module by means of scripts. A script consists of a series of commands, variables and programming logic that can perform various checks of the game state, change the state of the game, or return a result.

Scripts can be used in conversations to check whether a particular condition is true and to make various modifications to the module or campaign. Scripts are also used to process various events, and to control the behavior of creatures or objects during the game.

## Editing

The toolset includes a script editor that can be used to create and compile a script. There are two methods to create a script. From the File menu, select New then Script. (The keyboard equivalent is Alt+f-n-a.) Alternatively, from the pane that shows the various areas, conversations and scripts, select Scripts then right click the mouse and choose Add. There is also a menu of 'Add from Template' scripts to assist in the creation of some common script types.

To edit the script, select the script name from the list, right-click the mouse and choose Open.

The main panel is the editor itself, where you can create and modify the code. This language is similar to ANSI/C, so a familiarity with the latter is helpful. As always, make sure that non-comment statements within a function scope end in a semi-colon, all variables are declared before use, and the proper variable types are being passed to functions.

Underneath the editor is a results panel that shows the results of a compile or details a selected function or constant. The buttons along the top of the editor include a "Save & Compile" option. Clicking on this button will cause the toolset to attempt to compile the code. If there is an error, it will appear in the compile results field at the bottom of the editor.

To the right is a Script Assist panel showing the built-in functions, constants and template scripts. Selecting a

function or constant in this list will cause a brief description to appear below the editor. These are described in the second volume. The Script Assist can be hidden or displayed by toggling the Hide/Show Script Assist button at the top of the panel.

To close a script, right click on the Edit panel tab and select Close (or Ctrl+Shift+C).

## Variables

Variables are declared and used as in the C# language. In the toolset script editor, the standard variable types are printed in blue.

### Basic Types

The following basic data types are used by the built-in functions:

- A *boolean* is a variable that can have a value of TRUE or FALSE. The boolean is an *integer* that has a value of zero when it is FALSE, and non-zero when TRUE. Booleans are used to code 'if' or 'while' logic in a script. In the toolset code, it can be indicated by a 'b' prefix.
- An *integer* is a variable that can have a range of numerical values that do not have a fractional component. It is used to track values that change by whole increments, such as the amount of gold, a character's class level, the number of hit points remaining or a creature's Strength. In the toolset code, it is often indicated by a 'i' or 'n' prefix. Integers are signed 32-bit numbers and default to 0.
- A *float* is a 32-bit precision decimal value that can have a fractional component. It is often used to track physical scale within the game, such as the distance to an object or the scale of an enlarged object. A floating point constant can have an optional letter 'f' immediately after the decimal suffix.
- A *string* is a sequence of characters that can encode a name, tag or some operation. It can be defined by



## Editing

placing the characters inside double quotes. In the toolset code, it is often indicated by a 's' prefix. Strings include command overloading of the '+' operator, allowing strings to be appended. Within a script, double-quoted string values are printed in **dark red**.

A *const* statement can be used to create a constant declaration that can not be modified by the code, and it is placed before the variable declaration. Only an int, float or string can be declared as a const. For example:

```
const int MY_CONSTANT_VALUE = 12;
```

### Structured Types

There are also several standard variable types that encompass multiple fields and are used by specific function calls. These behave like a 'struct' in the C language, although I haven't been able to access the struct fields.

- An *event* will trigger a script to run.
- An *itemproperty* is an item power or limitation that is set via the 'Item Properties' field in the properties pane.
- A *location* is a position within the game. It includes coordinate information and the area. Locations correspond to the Position information in the Properties panel, but also include a facing and the Area identification.
- An *object* is a variable that represents some component within the game. It can be a creature, placeable, waypoint, item, and so forth. It is up to the script to determine whether the object is valid and has the appropriate tag or type.
- A *struct* can be declared that can include multiple arbitrary variables of different types. It is used much as in the C language, and can be passed as an input-only argument and returned as a result. See 'x2\_inc\_craft' for an example of how to create and use a struct.
- A *talent* is used to represent a feat, skill or spell.
- A *vector* describes the distance and direction of one *location* with respect to another *location*. Command overloading allow vectors to be added to and subtracted from each other, and they can be scaled by multiplication or division by a float value.

An *action* is a special type of variable that can not be declared. Instead it is used like a function pointer reference in calls such as AssignCommand that return a *void* value. Functions that begin with 'Action' represent types of activity that can be performed by a creature, and these can be passed as action arguments. Likewise, you can pass a script function as an action argument, as long as it returns a *void*.

### Initialization

Variables are initialized by default, but, as always, it is good practice to initialize them prior to use. This can be done on the same line as the variable declaration. Here is an example:

```
int bIsReady = TRUE;
int nCount = 10;
float fMaxRange = 20.5f;
vector vPosition = [ 0.0, 0.0, 0.0 ];
string sName = "My name is fred.";
object oCreature = GetObjectByTag(
    "c_blackdragon" );
```

Argument variables can have default initializers. These are defined in the function call definition, and the initialization means the argument does not be passed when the default value is acceptable. For example:

```
int MyFunction(
    int bVal = FALSE;
    object oTarget = OBJECT_SELF )
```

Note that if a value is passed for an initialized argument, then values must also be passed for all preceding arguments. Thus, in the above example, if an object is passed in the *oTarget* field, then a value must also be passed in the *bVal* field.

### Constants

The toolset includes a set of global constants that can be used within a script. These are typically in uppercase type with underscores as word separators. Constants have a fixed numerical value and are typically integers, but they can be any valid variable type. To see the built-in list of integer constants, select the Globals tab in the Script Assist area of the script editor. The Script Assist includes a Filter box that can be used to reduce the list to only those Globals that

## Editing

include the input value. For example, enter the following for a list of constants that include the substring '\_MODE\_':

```
MODE
```

Typically a 'const' is used to define a local constant declaration.

```
const float FX_SIZE = 15.0f;
```

Selected constants have a special meaning within a script:

- **OBJECT\_SELF** refers to the object that is executing the script.
- **OBJECT\_INVALID** is always an invalid object.
- **TRUE** is a binary true value, and **FALSE** is a binary false value.

## Flow Control

The toolset scripts provide C-like language functionality for logical flow control of the code.

An 'if' command will execute the code statement within a pair of curly braces if the statement within the parenthesis is true. (That is, the statement within the parentheses must have a non-zero value.) If the statement is false, the next statement in an 'else if' sequence is evaluated. If the previous statements were all false, then the final 'else' code is executed. The brackets are optional if there is only a single line of code.

```
if ( statement ) {  
    // Code here  
} else if ( statement ) {  
    // Code here  
} else {  
    // Code here  
}
```

A variant is the 'arithmetic if' operator, which evaluates an expression, then returns the result before the colon if true, or the second result if false.

```
int nAbsValue=(nTest > 0) ? nTest : -nTest;
```

A 'for' loop will repeatedly execute the code in the brackets as long as the 'test' is true. The 'initialization' action is executed before the first loop, and 'increment' is run at the end of each loop.

```
for ( initialization; test; increment ) {  
    // Code here
```

```
}
```

Typically this loop is executed by incrementing an integer then exiting the loop when the integer exceeds a value. For example, the following will run the code inside the curly braces ten times before exiting:

```
int i;  
for ( i = 0; i < 10; i++ ) {  
    // Code here  
}
```

A 'while' loop will repeatedly execute the code between the brackets as long as the 'statement' is true. A 'break' statement will exit from the inner-most 'for' or 'while' loop.

```
while ( statement ) {  
    // Code here  
}
```

A 'switch' statement will compare the 'value' to each of the 'case' values. If there is a match, the code will be executed up to the next 'break'. If there is no 'break' prior to the next 'case', then the execution will fall through and continue execute the code in the next case section. When none of the cases match, the code following the 'default' code is run.

```
switch ( value ) {  
    case value1:  
        // Code here  
        break;  
    case value2:  
        // Code here  
        break;  
    default:  
        // Code here  
        break;  
}
```

Functions are passed a set of zero or more parameters. The subroutine then performs a series of steps and, optionally, return a result. The result is passed back by a 'return' line with a value that must match the function declaration return type. For example:

```
object GetHeadGear(  
    object oCreature = OBJECT_SELF )  
{  
    if ( GetIsObjectValid( oCreature ) ) {  
        return GetItemInSlot(  
            INVENTORY_SLOT_HEAD, oCreature );
```

## Editing

```
}  
return OBJECT_INVALID;  
}
```

The only input parameter *oCreature* is an object; presumably a creature. The script first checks if the object is valid, then it returns the object that is in the head slot, if any. If the *oCreature* is an invalid object, or if there is no item in the head slot, then this function will return `OBJECT_INVALID`.

Functions are defined within the scope of a script, or via an include file. As in C++, the parameters can be given default values that are used when the call does not pass a value in that field. For example:

```
void MyTest(  
    int nValue,  
    object oPC=OBJECT_SELF )
```

When `MyTest` is called as follows:

```
int n = 0;  
MyTest( n );
```

then the second argument is set to `OBJECT_SELF`.

Variables can be defined that are local to the scope of a function and the 'if', 'for' or 'while' blocks, but not within a 'switch'.

## Compiling

The current script can be compiled by selecting the Save & Compile button at the top of the Edit panel, or by pressing f7. The compile will halt on the first error detected. These list the script name with the line number in parentheses, then the error message. It may not always be immediately clear from the message what problem has occurred.

Here are some of the error messages and the usual suspects. Many of these are obvious, but I am listing them for completeness.

### ARITHMETIC OPERATION HAS INVALID OPERANDS

- This can occur when a value in a string appending operation is not a string. Use one of the `String*` calls to convert the value to a string.

### DECLARATION DOES NOT MATCH PARAMETERS

- Check the arguments to the function call and see if the

number of arguments and their types match the declaration. (Double-click on a built-in function to see the declaration.) Non-initialized arguments must be provided.

### FUNCTION MAIN() MUST HAVE A VOID RETURN VALUE

- Set the return value of `main()` to void.

### INVALID DECLARATION TYPE

- There is an extra closing bracket before this line.

### MISMATCHED TYPES

- Setting the value of a variable to a different type, such as assigning a float value to an integer.

### MULTIPLE CASE CONSTANT STATEMENTS WITHIN SWITCH

- At least two case statements within the same switch are checking the same constant value.

### NO SEMICOLON AFTER EXPRESSION

- Check earlier in the script for a non-comment line without a closing semicolon.

### NO SEMICOLON AFTER STRUCTURE

- A structure definition must end with a semi-colon.

### NON OPTIONAL PARAMETER CANNOT FOLLOW OPTIONAL PARAMETER

- This applies to function arguments that include a parameter that has a default value, followed by a parameter without a default value. All parameters with default settings should be at the end of the list.

### NOT ALL CONTROL PATHS RETURN A VALUE

- The end of the function does not return a value.

### PARSING VARIABLE LIST

- A variable initialization may be missing a semicolon.
- A variable initialization may be malformed.
- A subroutine name may be invalid.

### UNDEFINED IDENTIFIER

- A function call may be missing or spelled incorrectly.

### UNKNOWN STATE IN COMPILER

- A function may be missing a closing parenthesis, a nested parenthesis or a comma.
- Attempted to declare an *action* variable.
- Attempting a C-style type cast of a constant.
- A comment tag `*/` exists without a preceding `/*`.

## Editing

### UNDETERMINED STRING CONSTANT

- A string is missing a double-quote character.

### UNDEFINED FIELD IN STRUCTURE

- A structure variable name is being used that is not defined within the structure.

### UNEXPECTED END COMPOUND STATEMENT

- Missing a closing bracket.

### VARIABLE ALREADY USED WITHIN SCOPE

- A variable has been defined twice within the scope.

### VARIABLE DEFINED WITHOUT TYPE

- A variable being used that does not have a defined type within the current scope. It should be declared prior to the first use.
- A constant value is not spelled correctly.

## Debugging

When your module is running with the Toolset in the background, you can activate the console interface by pressing the back-tick/tilde key (`/~). This will cause a shaded area to appear at the top of the display, along with a '#' prompt where commands can be entered. To exit the interface, press the back-tick/tilde key again.

## Commands

You must be in debug mode in order to run commands at the console prompt. This is activated by typing the following command at the prompt:

```
DebugMode 1
```

After the console interface is activated, a list of available commands can be displayed by typing:

```
command [filter]
```

where filter is a selection pattern. If no filter pattern is supplied, all of the commands are displayed. Entering a single letter will show all commands that begin with that letter (whether upper or lower case.) However, a word filter is case sensitive. Thus, you will need to use 'command Print' to find the print commands.

The console commands are case sensitive. Many of the commands modify the game graphics, cause internal changes to the current game or print information that may be useful for debugging (such as the values of variables). Some of the commands toggle features on and off, so that running the same command will switch to the opposite setting.

Here is a brief description of some commands:

- **Aabbboxes** – toggle the display of gray bounding boxes around each of the visible contents in the area. In the toolset, these boxes are seen (in yellow) by selecting 'Bounding Boxes' from the Collision popup menu.
- **Animation** – toggle the display of animation for dynamic area contents, such as creatures.
- **BugReport** – this will open up a bug report text file.
- **C2** or **C3** – toggle the display of C2 and C3 data, as

## Debugging

activated in the toolset by selecting 'C2 Data' and 'C3 Data' from the Collision popup menu.

- **Capsules** – toggle the display of some creatures inside white spheres.
- **CeilingMode** *mode* -- this sets the display mode for interior ceilings. A mode of 0 will hide the ceiling when it blocks the view of the player; 1 will always block the camera, and 2 will prevent ceilings from being drawn. Mode 1 is the same as selecting character mode.
- **Daynight** – toggle between day and night.
- **Dropshadows** *state* – setting the *state* to zero will turn off drop shadows. Setting it to 1 will turn them back on.
- **Envshadows** *state* – setting the *state* to zero will turn off shadows from static placeables. Setting it to 1 will turn them back on.
- **Farshadows** *state* – setting the *state* to zero will turn on shadows from distant objects. Setting it to 1 will turn them back off.
- **Gfxoptions** – this will open a dialog box that allows you to set various graphics options, including toggling the rain mode.
- **GiveFeat** *ID* – adds the feat identified by *ID* to the current PC, or else all feats if no *ID* is specified. The *ID* is a row number in the 'feats.2da' file.
- **GiveItem** *resname* – gives a copy of the item with the Resource Name *resname* to the current PC.
- **GiveSpell** *ID* – this attempts to cause the current PC to memorize the spell identified by *ID*. The *ID* is a row number in the 'spells.2da' file. If the PC is of the wrong class to cast the spell, then an error results. However, there is no check for the required caster level or number of spells allowed. Thus the 6th-level *blade barrier* spell, with an ID of 5, can be given to a 1st-level cleric.
- **GiveXP** *amount* – awards the current PC an *amount* of XP. If *amount* is not specified, the PC is awarded enough to reach the next level.
- **Hookpoints** – toggle the display of hook points on weapons.
- **Lights** – toggle the display of light radius spheres.
- **Loc** – prints the PC's current {x, y, z} vector position.
- **MemStats** – prints a dynamic table of memory usage data.
- **Ooboxes** or **obb\_all** – toggle the display of selection boxes around every content and tile in the area.
- **obb\_cdoor** – toggle the display of the selection boxes around doors and door frames.
- **obb\_water** – toggle the display of the game's rendering rectangles around water planes.
- **Paths** – this toggles the display of a yellow cross under each creature and a line pointing in the direction each creature is facing. This line turns as the creature changes facing.
- **Polymorph** *ID* – causes the current PC to polymorph into the creature with object identifier *ID*. This *ID* is a row number in the 'polymorph.2da' file. See unpolymorph.
- **PrintCreatures** – prints the name, tag and object ID of creatures in the area.
- **PrintFeats** *target* – lists the feats of the creature with object ID *target*. If *target* doesn't exist, this will print the feats of the PC.
- **PrintLevelStats** – prints the class, levels and remaining skill points for the PC.
- **PrintLocalVars** *target* – lists the local variables assigned to the *target*. This may print the output to a file under a system-hidden folder called AppData in your user directory.
- **PrintPerception** – prints a report about perception from the perspective of the PC and the area creatures.
- **PrintGlobalVars** – lists the global variables.
- **PrintRepository** *target* – lists equipped items and inventory of the *target*.
- **PrintReputation** *target* – prints a table of reputations with respect to the *target*. The table shows reputation score, whether it is in the creature's party, the faction ID, object ID and name/tag.
- **PrintScripts** *target* – prints the script names for the *target* at the console.



## Debugging

- **Rain state** – set the rain *state*. A state of zero turns off the rain.
- **RemoveFeat ID** – remove the feat identified by *ID* from the current PC. The *ID* is a row number in the 'feats.2da' file.
- **Resourcestats** – prints the video memory statistics.
- **rs** – See RunScript.
- **RunScript script( arg1, arg2, ... )** – This will run the script named *script*, passing in a comma-separated list of arguments: *arg1*, *arg2*, and so forth. This can be used, for example, to run various action scripts, ga\*. Thus, the following command will give the PC the *toughness* feat (which is listed as #40 in 'feat.2da'):  

```
# RunScript ga_give_feat( "", 40, 0, 0 )
```
- **Sceneintens [value]** – when run without an argument, this will print the current value of the bloom scene intensity. When a floating point value is passed for the *value*, this will change the bloom scene intensity to match. Thus: Sceneintens 0.1.
- **Shadows state** – setting the *state* to zero will turn off shadows. Setting it to 1 will turn them back on.
- **Skels** – toggle the display of creature skeletons. This is the same as selecting the Skeletons button in the toolset.
- **Sky** – toggle the display of the sky.
- **Stats** – toggle the display of summary statistics in the lower left corner of the display.
- **Surface** or **Surfaceonly** – toggle the display of the baked surface mesh.
- **TakeDamage amount target** – apply *amount* of hit point damage to *target*.
- **Textborder** – toggle outline boxes around the player interface graphics, such as the chat window.
- **Trees** – toggle the rendering of trees.
- **Unpolymorph** – return a polymorphed creature to its normal appearance.
- **Wami** – "where am i". This is the same as the Loc command.
- **Wireframe** – enables wireframe mode. This is similar to selecting the Wireframe button in the

toolset. *I haven't found a method to turn this off, other than to restart the game.*

The commands are not case sensitive. Help on a some commands is obtained with the 'help' command, followed by the command name. The up/down arrow keys can be used to scroll through previously run debug commands.

## Script Messages

In order to display debugging messages from scripts, you will need to configure a setting in your account's copy of the *nwn2player.ini* file. On Windows, this file should be located under the documents folder of your user directory within a sub-folder called Neverwinter Nights 2. If you can't find it, try a search on *nwn2player*.

There are two variables of interest located at the bottom:

```
[Server Options]
Scripts Print To Log=0
Scripts Print To Screen=0
```

To display the messages on the screen, set the second name-value pair equal to 1. Save the file and start the toolset.

The debugging messages will display after typing the following at the console interface:

```
# debugmode 1
# debugtext
```

Any messages generated by a DebugPostScript function should now appear on the upper left of your display.

The *ginc\_debug* include file has several convenient routines that can be used to print messages to the screen from your scripts. For example:

```
#include "ginc_debug"
...
PrettyDebug( "Display a debug message." );
```

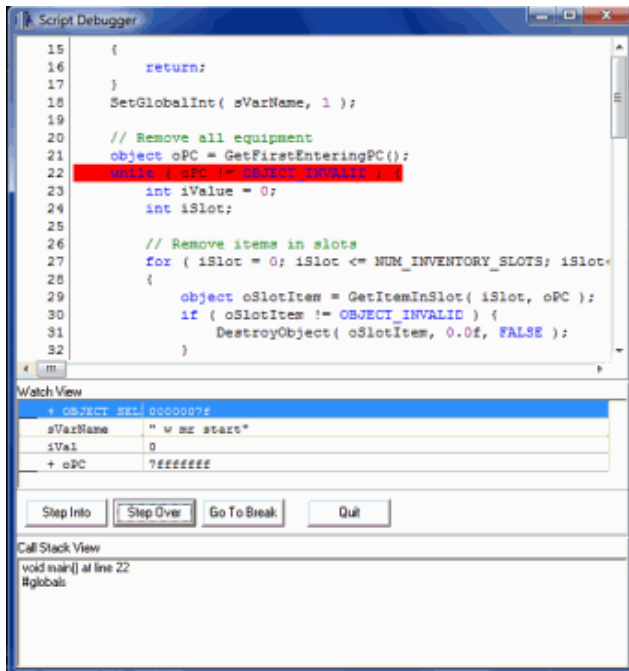
Many of the built-in scripts will print debug messages that you will see while the game is running.

## Script Debugger

As the name implies, the script debugger is a tool that allows you to step through a script and check if it is working as expected. Use of the script debugger requires that scripts be compiled with debug information. To do this, choose 'Options...' item from the View menu, select 'Script'

## Debugging

and change 'GenerateDebugInformation' to true.



*The script debugger interface*

The script debugger can be launched from within a script by means of the `SpawnScriptDebugger()` call. To run the debugger, you will need to do the following:

- In the Utils folder under the install directory, run the DebugServer tool. This will create a small window on the screen.
- The Neverwinter Nights 2 game needs to be run in windows mode. In your Neverwinter Nights 2 document folder, edit the `nwn2.ini` file and change to 'Fullscreen=0'. Note that when the game is in windows mode you'll have to use the arrow keys to turn the view.

For the script you want to debug, place the call to spawn the script debugger just prior to the code you want to debug, then recompile the script. The debugger interface will appear whenever the script is being run by the game.

*Don't forget to remove the `SpawnScriptDebugger()` call after you have finished debugging.*

## Examples

To create your first scripts, it is helpful to examine the various script templates and to peruse the scripting reference in the toolset's build-in help. Additional examples can be found in volume III.

Here is some trivial sample code to demonstrate the notation:

```
#include "file";

/* My sample subroutine */
int my_test( object oPC ) {
    int n = 0;

    /* Use a built-in function to test if
     * the argument is a valid object.
     */
    if ( GetIsObjectValid( oPC ) ) {
        int i; // Define index before 'for'
        for ( i = 0; i < 10; i++ ) {
            // Do something 10 times
        }
    }

    /* Call GetName to get object Local Name
     * then use overloaded string comparison.
     */
    string sMsg = GetName( oPC );
    if ( sMsg == "me" ) {
        // This object is "me" so do something
    }

    // Sample switch statement
    n = GetNumActions( oPC );
    switch ( n ) {
        case 0:
            // Idle
            break;
        case 1:
            // Single action
            break;
        default:
            // Anything else
            break;
    }
}
```

## Examples

```
}  
return n;  
}
```

The 'if' and 'while' blocks allow variables to be defined in scope, but this is not true of a switch statement. The index variable must be defined before it is used in a 'for' statement.

### First-Next Pairs

The built-in functions include a number of First-Next pairs that will cycle through a list of objects of a particular type. Each time the 'First' function is called, it will reset the list and return the first object matching the type. The subsequent calls to the 'Next' function will retrieve the next object in the list. By always checking whether the returned object is valid, you can use this sequence to cycle through all objects in the list using a while loop.

For example, the following loop will cycle through each of the properties on a valid item *oItem*, returning true if a property provides darkvision.

```
int GetItemHasDarkvision( object oItem )  
{  
    // Initialize the item property list  
    itemproperty iProp =  
        GetFirstItemProperty( oItem );  
  
    // Cycle through the valid properties  
    while ( GetIsItemPropertyValid(iProp) ) {  
        // Check for darkvision  
        int nIPT = GetItemPropertyType( iProp );  
        if (nIPT == ITEM_PROPERTY_DARKVISION)  
            return TRUE;  
  
        // Get the next item property  
        iProp = GetNextItemProperty( oItem );  
    }  
    return FALSE;  
}
```

### Inflict Lightning Damage

Suppose I want the object running the script to inflict some number of six-sided dice of lightning damage to a target creature, but the creature is allowed a Reflex saving throw and can Evade, per the feat. In the sample code

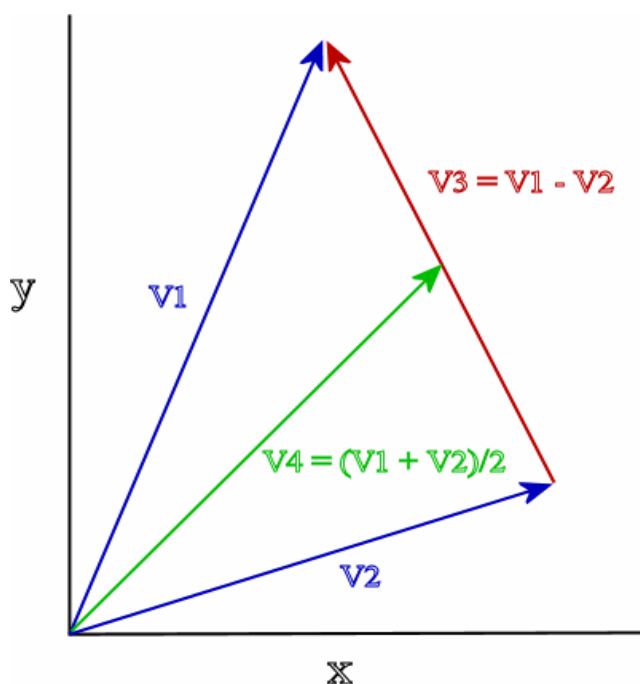
below, the damage is randomly generated using a  $d6(nDice)$  call plus *nModifier*. The damage is reduced if a save is successful against the supplied difficulty class *nDC*. Finally, the damage is applied to the target as an electrical type effect.

```
void ApplyElectricalDamage(  
    object oTarget, int nDice = 6,  
    int nModifier = 0, int nDC = 14 )  
{  
    // Get the base amount of damage  
    int nDamage = d6( nDice ) + nModifier;  
  
    // Reduce on a reflex save against DC  
    nDamage = GetReflexAdjustedDamage(  
        nDamage, oTarget, nDC,  
        SAVING_THROW_TYPE_ELECTRICITY );  
  
    // Apply positive damage as an effect  
    if ( nDamage > 0 ) {  
        effect eDamage = EffectDamage( nDamage,  
            DAMAGE_TYPE_ELECTRICAL,  
            DAMAGE_POWER_ENERGY );  
        ApplyEffectToObject(  
            DURATION_TYPE_INSTANT, eDamage,  
            oTarget );  
    }  
}
```

### Moving to the Midpoint Between Objects

The following subroutine causes the creature *oCreature* to move to the midpoint between the two placeable objects passed as arguments. For simplicity it skips any checks for object validity. To find the midpoint, the positions of the objects are determined. This will provide a vector to each object from the area coordinates origin, as shown by V1 and V2 in the following illustration.

## Examples



*Illustration of the vector math used below*

The midpoint between the two objects is then given by their average value (V4), as shown in the illustration. The final facing of the creature is set to a direction perpendicular to the line between the two objects by computing the vector (V3) between them, then converting the vector to an angle and adjusting it by 90°.

```
void MoveToMidpoint(
    object oPlaceable1, object oPlaceable2,
    object oCreature = OBJECT_SELF,
    int bRun = FALSE )
{
    // Vector math
    vector V1 = GetPosition( oPlaceable1 );
    vector V2 = GetPosition( oPlaceable2 );
    vector V3 = V1 - V2;
    vector V4 = ( V1 + V2 ) / 2.0f;
    float fFace = VectorToAngle( V3 ) + 90.0f;

    // Get the location of the midpoint vector
    object oArea = GetArea( oCreature );
    location locMidpoint = Location(
        oArea, V4, fFace );

    // Move creature to midpoint
    AssignCommand( oCreature,
```

```
ActionMoveToLocation(
    locMidpoint, bRun );
}
```

### Visual Effect

In the following example, this heartbeat script will create a small fireball at an Ipoint every 6+1d6 seconds. This uses the effects referenced by the VFX\_... constants. The usable visual effects can be found in the 'visualeffects.2da' file. Look for a VFX\_... constant in the Label column and then see if it has a '.sef' file allocated. (Ignore the rows with the 'fx\_question\_fountain.sef' file.)

```
void main()
{
    // Create fireball after 1d6 seconds
    effect eEffect = EffectVisualEffect(
        VFX_HIT_AOE_FIRE );
    DelayCommand( IntToFloat( Random( 6 ) ),
        ApplyEffectToObject(
            DURATION_TYPE_INSTANT, eEffect,
            OBJECT_SELF ) );
}
```

There are a set of special effect files that can be applied at a location. These have a '.sef' suffix and are located under the Data\NWN2\_VFX... folders in the game's install folder. The various effects are also available as selections on the 'Appearance (visual effect)' pick for various blueprints, and are briefly described in the Effects Files chapter of the second volume. Effects for these effect files can be accessed using the EffectNWN2SpecialEffectFile call. A sequence of effects can be strung together by means of DelayCommand calls with the appropriate durations.

In the following example, a local teleportation effect is applied to the oTarget's location. The target must undergo a jump after 1.5f seconds to synchronize with this effect. (See the JumpPartyToArea command for a group of PCs.)

```
// Create a teleportation audio/visual effect
void ApplyTeleportEffect( object oTarget )
{
    if ( ! GetIsObjectValid( oTarget ) )
        return;

    // Generate the effects
    effect eStart =
```

## Examples

```
EffectNWN2SpecialEffectFile(
    "fx_ritual_replenish.sef" );
effect eFinish =
    EffectNWN2SpecialEffectFile(
        "fx_teleport_new.sef" );
effect eVanish =
    EffectNWN2SpecialEffectFile(
        "fx_ethereal.sef" );

// Apply effects to the target's location
location locTarget = GetLocation(
    oTarget );
ApplyEffectAtLocation(
    DURATION_TYPE_INSTANT,
    eStart, locTarget );
DelayCommand( 0.7f, ApplyEffectAtLocation(
    DURATION_TYPE_INSTANT,
    eFinish, locTarget ) );
DelayCommand( 1.6f, ApplyEffectToObject(
    DURATION_TYPE_TEMPORARY,
    eVanish, oTarget, 4.0f ) );
}
```

### Allow Limited Rest

Suppose you only want to allow the party to rest in certain locations within an area, thereby making recovery of spells and hit points more challenging. A modified version of the 'x1\_playerrest' script, in combination with the trigger tagged 'X0\_SAFEREST', can be used to define these safe locations. First you can make a copy of the 'x1\_playerrest' script and customize it for your module; you could call it, say, 'on\_playerrest'. Inside this script is a subroutine called 'NotSafeToRest' that returns true if rest should not be permitted in most of an area.

In your script copy, the 'NotSafeToRest' routine can be modified to the following simple form:

```
int NotSafeToRest( object oPC )
{
    return GetLocalInt( GetArea(oPC),
        "bLimitedRest" );
}
```

After this modified script is placed in the Module's 'On Player Rest Script' property, the script's 'main' routine will be called whenever the player attempts to rest. If an area has

the boolean variable 'bLimitedRest' set to 1 in its 'Variables' property, the 'NotSafeToRest' routine will return 1 and the script will check the 'NotSafeOnRest' routine to see if the player is in a safe situation. If the situation is found to be unsafe, the rest will be aborted and a floating string message will appear above the PC.

Per the comments for this blueprint, the 'X0\_SAFEREST' trigger must be painted so as to include the safe rest location plus a door leading into the location. The 'NotSafeOnRest' routine will then check for the nearest object with the tag 'X0\_SAFEREST', which is the safe rest trigger. If the PC is within the trigger area and the trigger includes a closed door, then the PC is allowed to rest. Otherwise the rest is aborted and a message is printed.

Note that this method only works in areas that have the 'No Resting Allowed' property set to *false*. If this property is true, the 'On Player Rest Script' is not even run.

Once the safe location is discovered by the player, you may want a script to mark the safe room with a 'Safe Waypoint' map note and broadcast a message indicating it is a safe place to rest.<sup>58</sup> This script could, for example, be run from the trigger's 'On Enter Script' property the first time the party enters the room. The note will then show up on the player's map and the party can return to the room wherever they want to rest. You may even want to set the waypoint Color property to green as an indication of safety.

### Varying daylight sources

If you want interior lights to only be active during the daylight, you can create a script that will toggle the state of the light placeables using the SetLightActive() call and cycle through all the window light sources. If you use light sources to simulate window illumination, you should give your lights unique tags so that they can be readily obtained using GetObjectByTag() calls.

Typically you would call the light variation script from an interior area's On Client Enter Script property field. Running the script from the On Heartbeat Script field would consume more system resources, and besides, suddenly

---

<sup>58</sup> Here's a StrRef for a floating string message: #125778: "If we shut the door we could rest here safely, I wager."



## Examples

turning the window lights on or off wouldn't be very realistic. (Conceivably you could, however, use a group of multiple lights and transition between them over the course of the day.)

### Item Scripts

The Module properties has a set of fields for event handler scripts that apply across all areas. Several of these fields have default scripts that run whenever certain categories of events occur. Five of these event types are related to items:

Module Event Handler Field	Requires Script
On Acquire Item Script	x2_mod_def_aqu
On Activate Item Script	x2_mod_def_act
On Player Equip Item Script	x2_mod_def_equ
On Player Unequip Item Script	x2_mod_def_unequ
On Unacquire Item Script	x2_mod_def_unaqu

If the listed module scripts are retained in their fields and one or both of the following switches are set to true in the 'On Module Load Script' script x2\_mod\_def\_load:

```
MODULE_SWITCH_ENABLE_TAGBASED_SCRIPTS
MODULE_SWITCH_ENABLE_SEPARATE_ITEM_SCRIPTS
```

then items can have appropriately named scripts that will process these events for PCs. (Note that these scripts will *not* run for NPCs, even if they are members of the party.) These scripts will be launched whenever the corresponding module events apply to that item. Thus an item's equip item script will be run whenever that item is equipped by a PC.

Item scripts allow items to be created that have powers and behavior that aren't covered by the standard Properties. There are two methods of naming scripts that will be run when item events occur.

#### Tag-based Script

This method allows all of the code related to an item to be placed in a single script. This script must have the same name as the item tag, although the script name can be in lower case. This script should call the following routine from the x2\_inc\_switches include file:

```
int GetUserDefinedItemEventNumber()
```

This will return one of the X2\_ITEM\_EVENT\_\* constants defined in the same include file. Each applicable event can then be processed by the script using a switch statement. An example of such a script is 'x2\_it\_example'. In addition to module events, this script can also include checks for

## Item Scripts

unique power properties of the item.

Note that an item with charges will disappear from the character's inventory when the last charge is used. However, this does not cause the On Unacquire Item script or the On Activate Item script to fire. The same is true of the DestroyObject function call.

### Separate Item Scripts

You can have separate scripts for each of the five module events. The script names must have a prefix of “i\_” (for item), followed by the item tag, then by a suffix that depends on the event type. Script templates are available in the A/C/S panel. Just right-click, select "Add from Template", followed by the appropriate Item script. Here are the available item scripts (as described in the Script Naming Conventions section of the built-in help) for an item with tag *item\_tag*:

Item script name	Executed when...
i_item_tag_ac	item is activated
i_item_tag_aq	item is acquired by party
i_item_tag_ua	item is unacquired by party
i_item_tag_eq	item is equipped
i_item_tag_ue	item is unequipped
i_item_tag_hc	weapon item hits target <sup>59</sup>
i_item_tag_ci	spell cast at item

The suffixes are defined by the:

```
SCRIPT_EXTENSION_ITEM_EVENT_*
```

constants found in the x2\_inc\_switches script.

### Example

Suppose I want to create a ring that will teleport the party to an “extradimensional” chamber where they can rest up in safety? First I create a Ring blueprint and give it the tag 'ring\_pocket\_plane'. Next, I modify the Item Properties of the ring to add a 'Cast Spell' property of type 'Unique Power Self Only'. I select the property and change the CostValue allow use once each day. This power can now be activated by right-clicking on the ring, selecting 'Activate Item (Self)'

and left-clicking a target character, which I will limit to members of the party via the script.

Now I click the Scripts tab in the A/C/S panel, right-click to select 'Add From Template' then pick 'Item Activate'. I rename the resulting script to 'i\_ring\_pocket\_plane\_ac'.

In the script I execute a JumpPartyToArea call to transport the party to chamber. However, I also want to be able to return to the original location. To do this, the script creates a unique waypoint at the starting point; taking care to clean up any old instances first. In the pocket plane I add a door that will transition the party back to the created unique waypoint.

---

<sup>59</sup> An "\_hc" script is run when an item has an "On Hit Cast Spell: Unique Power (onhit)" Item Property.

## References

## References

- *Bioware Aurora Engine, Common Game GFF Structures*. Bioware Corp.
- Jackyo123 (2007). *Cutscene Tutorial Part 1 – Basic Cutscenes*.
- Grimlar. *Introduction to Tag Based Scripting*.
- Sunjammer (2009). *Moveable Placeables*.
- Eligio Sacatega (2003). *Neverwinter Nights: Custom Content Guide v3.0*.
- NWN2 Toolset Beta Testing Community (2006). *Neverwinter Nights 2 Toolset: Tips and Tricks*.
- Koroush Ghazi. (2009) *Neverwinter Nights 2 Tweak Guide: Advanced Tweaking*.
- *NWN Lexicon*.
- Lance Botelle. (2008) *NWN2 XML & GUI Coding for Beginners*.
- Zarathustra217. *Placeable Walkmesh Helper*.
- James Hastings-Trew. [\*Reproducing Real World Light\*](#). JHT's Planetary Pixel Emporium.
- David Gaider. *Scripting Tutorial*.
- Nathaniel Chapman (2008). *Storm of Zehir Community Tutorial: Overland Map*. Obsidian Entertainment.
- Khalidine (2008). *Toolset FAQ*. Bioware Community.
- Chris O'Sullivan (2008). *Toolset Manual, version 1.5*.
- Sunjammer (2007). *World Map Guide v1.01*.

## Revision History

### Revision History

Date	Version	Description
05/25/2009	1.0	Initial version.
06/03/2009	1.1	Light/shadow exception; more properties documented; applied suggested fixes and additions; included internal hyperlinks to chapters and sections.
06/13/2009	1.2	Minor copy edits for flow; added a plug-ins section; included layout/camouflage suggestions for interior areas; door transition; corrections.
12/07/2009	4th ed.	Many small copy edits for clarity. Additions for allowing limited rest, viewing animations, scaling doors, making a concealed passage, adding gold pieces to containers and combining effects for producing realistic light sources. Included additional remarks on stores.
07/28/2010	5th ed.	Additions for crafting items, recipe books, the World Map Editor and special abilities. More information on lights, sounds, creature scripts, the debugging console, and area appearance.
09/22/2011	6th ed.	Additions for intelligent weapons, lights, special abilities, overland map, and trees. Merged custom items section into special abilities. Reorganized the magic item lists. Extensive editing. More example scripts.
09/01/2013	6.1 ed.	Removed incorrect paragraph about scaling doors; added more sample scripts; added a short section about one-liners.