Grim Dawn Modding Tools

Grim Dawn features a powerful set of modding tools, available for free to all owners of the game. Because these are the same tools we used to create the game itself, they offer an exceptional degree of freedom for you to edit and create your very own content.

Modding can be a challenging, but rewarding endeavor. The freedom provided by our tools does mean that there is a steeper learning curve if you wish to tackle some of its more powerful functions. Not all tasks are created equal. For example, editing an item's stats is considerably easier than scripting an event where monsters attack a village.

To aid you in the creation of your first mod, we have created these tutorials, which go over the basics of using Grim Dawn's modding tools and get you started on your way to creating your own content!

Note that these tutorials are not fully comprehensive. They do not touch upon the finest nuances that may require an expert user and will not teach you how to make game assets such as models, animations and special effects.

We here at Crate Entertainment are excited to see everything that our players will come up with. Welcome to the Grim Dawn modding tools!

Tutorial List:

- Tutorial 01 - Asset Manager, how to create new mods and manage mod assets
- Tutorial 02 - World Editor, the basic functions for building and editing environments
- Tutorial 03 - Quests, the creation of quests (main story and side content)
- Tutorial 04 - Conversations, the creation of NPC dialogue (which can grant quests)
- Tutorial 05 - Monsters, the creation of basic enemies and boss monsters
- Tutorial 06 - Equipment, the creation of basic loot and loot tables
- Tutorial 07 - Scripting, an advanced tutorial on Lua scripting in Grim Dawn

ModdingTutorial Reference Mod

Your installation of Grim Dawn comes with a pre-made test mod called ModdingTutorial. This mod serves as a frame of reference for a variety of features and gameplay elements, such as questing, Lua scripting and monsters. All of the source files for this mod are included, so you can make changes and see how they impact the mod.

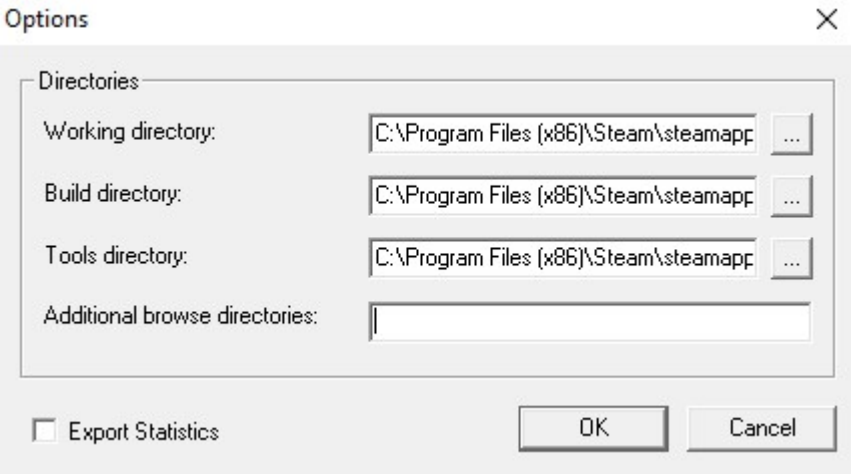
To check out the tutorial mod, simply extract ModdingTutorial.zip (located in the game installation folder) into the "/mods" directory. Note that if you change the Working directory (see Asset Manager tutorial), the location you need to place the source files will be different.

Tutorial 01: Asset Manager

This tutorial covers the Asset Manager and creating a new mod. Before you can work on a mod, you will need to launch the Asset Manager to perform **First-Time Setup**. You will also want to **Extract Core Game Files** so that you can access the game's default templates (for the creation of new files) and database (for referencing existing game files).

First-Time Setup

When you start up the Asset Manager for the first time, you will be asked to create a Working and Build directory. These are set to the game's installation directory by default so that the World Editor and other mod tools can load the default game assets. If you choose to change the Build directory, note that your tools will not load the game's assets and will instead load only assets you create in your mod.



Extracting Core Game Files

Chances are that you may want to make a mod that alters existing game data rather than creating something entirely new. Or perhaps you just want to take a peek under the hood and see how certain things in Grim Dawn were made. To facilitate this process, the Asset Manager is capable of extracting all of the game's database, template and asset files for reference purposes. Template files will be critical to creating new game records, such as items, monsters and skills. Database records and templates are extracted to /database while assets are extracted to /resources.

To extract game files, go to Tools -> Extract Game Files and select the Grim Dawn installation directory (default: C:\Program Files (x86)\Steam\steamapps\common\Grim Dawn\). This will overwrite any existing files in the /database and /resources folders, so do not run this process if you have files in there you are working on. This will rip all of the records, assets and template files from the core game archives.

Note: this requires at least 5gb of additional storage space on your hard drive as you are extracting compressed game files. This process may take several minutes during which the Asset Manager may become non-responsive.

If these files are placed within your mod following the same folder structure as the base game (ex. database/records/items/gearhead/a01_head001.dbr), then any changes you make will overwrite the original file while you are running your mod.

Basics

The Asset Manager is where all your mod files will be stored and organized. They are organized under 3 Tabs: Sources, Assets and Database.

Sources is where core game files are stored. These are anything from art assets (models, animations, textures) to sounds to quests and NPC conversations.

Assets is where various source files are referenced in a format recognized by the game so that they can be assigned to Database files (ex. a model may have a .msh asset which a decoration uses).

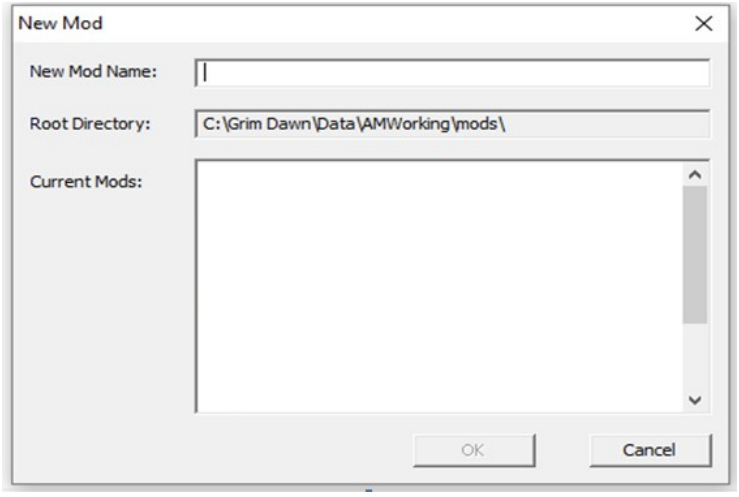
Database is where files that define gameplay elements are stored. These are things such as monsters, NPCs, level art, equipment and skills.

Creating a New Mod

Once you are comfortable enough with the tutorial mod, you will likely want to create your own. To do this, you must first create a new Mod in the Asset Manager.

Go to Mod -> New

Enter a name for your Mod and press OK.



The mod will be created with several important directories already added for you. These directories should not be deleted/renamed if you like your mod working. The folder names are pretty self-explanatory.

Sources/Maps: this directory is where all the world map data should be saved. You can create sub-directories here for organization purposes.

Sources/Quests: this directory is where all quest files should be saved. You can create sub-directories here for organization purposes.

Sources/Scripts: this directory is where main.lua needs to be placed as an entry point. All other script files can go wherever you wish.

Sources/Text_EN: this directory is where all string files (.txt) should be saved. This is how you define the names of objects in the game world (ex. tag_Sword01=Sharpened Sword)

How you organize other types of files is up to you.

Adding New Files and Folders

To create a new folder, or sub-folder, right-click in the folder list and select Create Directory. Name your folder as desired.

When in the Database tab, you can also create new Database Records (.dbr). These records come without a template, which must be assigned in order to edit it further.

Adding External Files to the Mod

Now that you have a brand new mod, you probably want to put something in it (a reasonable request). Since your mod starts out empty, you will begin by creating files outside of the Asset Manager that then need to be added to your mod. Once you've established a file base, you can copy-paste files within the Asset Manager if you want to create variants of things you've created before.

Source and Database Files

If you created Source or Database Files outside of the Asset Manager (ex. quest, conversation, art files, monster database record (.dbr)), if you place them inside of a directory contained within your mod, simply selecting another folder inside of the Asset Manager and then return to the original directory to refresh it. Note that you will need to restart the Asset Manager if you add a new folder outside of it.

Asset Files

Asset files are created by right-clicking Source files in the Asset Manager and selecting Auto-Create Asset. Depending on the file type, you may have to select a type of Asset to create.

Source files that require an Asset:

- Conversations
- Quests
- Art files such as textures, models and animations
- Special Effects
- Sounds
- Lua scripts
- The World (.wrl)

3ds Max Exporter

Grim Dawn comes with .dlo files for the following 32-bit versions of 3ds Max: 2009, 2011 and 2012, allowing you to create and import your own meshes and animations into the game. The files are available in the installation directory for the game.

Compiling a Mod

Once you've made all the desired changes and additions to your mod, it is time to test it!

Simply go to Build -> Build in the Asset Manager to compile your changes. Make sure you do not have any editing tools open, such as the World Editor. Once the Asset Manager is finished, you can launch the game and try out your mod.

Enjoy your hard work!

Packaging a Mod for Sharing with the World

Sooner or later, a time may come when you want to share your hard work with the rest of the world. In order to do this, you will need to package up your mod build files. The files other players need to play your mod are:

- /steamapps/common/Grim Dawn/mods/ModName/database.arz
- /steamapps/common/Grim Dawn/mods/ModName/resources/*.arc

Preserve the file structure in a .zip (or other compressed format). Other players must then extract your files in /steamapps/common/Grim Dawn/mods/. Remember to include all of the .arc files.

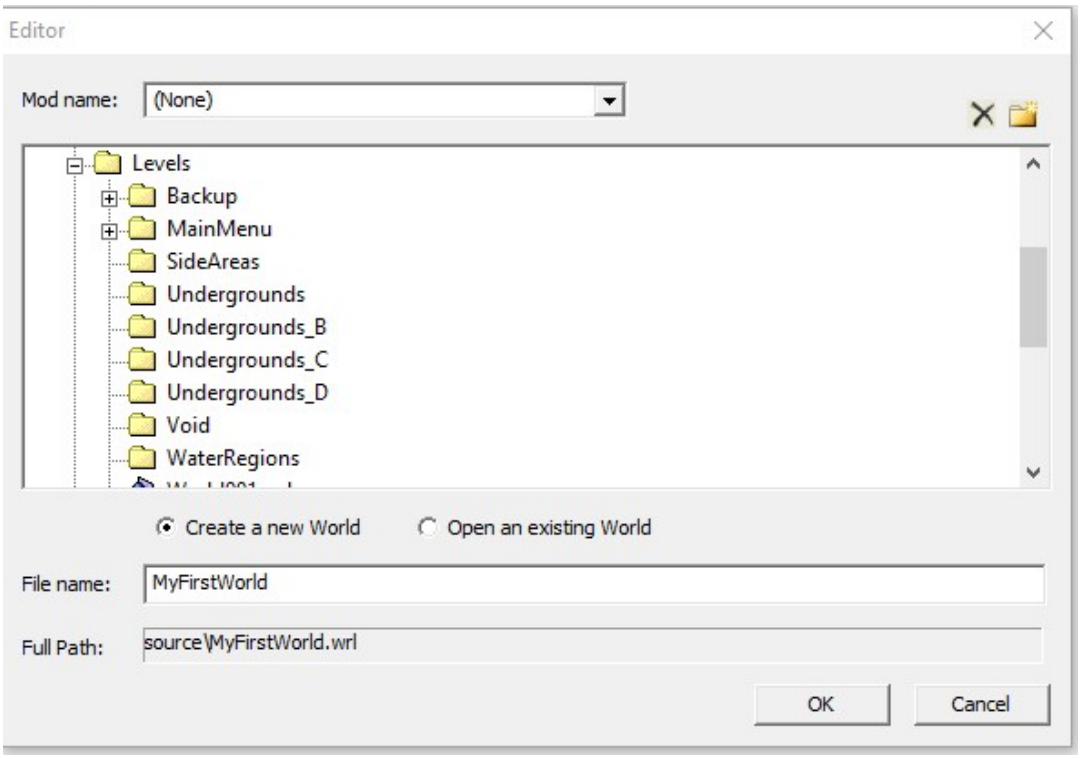
All the other files are source files for your mod and are only required to actually work on the mod and its contents. They do not need to be shared unless you want to share your work with a team of modders or simply wish for your work to be open to editing by the public.

Tutorial 02: World Editor

This tutorial covers the World Editor and its many functions which you will need to learn in order to create your own levels, fill them with content and bring your ideas to life.

Creating a New World

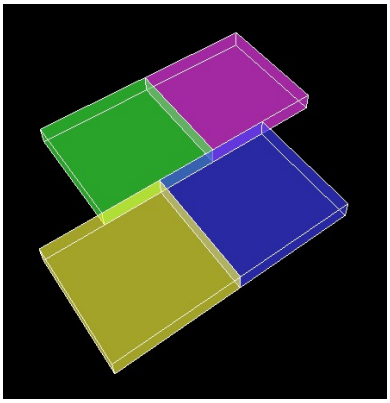
Before you can create levels, you must have a world to put them in. When you first launch the editor, you will have the option to do so. We recommend saving it in Sources/Levels/



Once you name your world, you will be presented with the infinite blackness of the world editor, because there are no regions in it yet! To add a region, go to Region -> Add New Terrain

You will see a similar window to creating your World (again, we recommend placing the levels in Sources/Levels/), but here you can also set the size of the region.

For optimal performance, we recommend an off-set grid system of 128x128 regions, which looks like this once you've added a few:

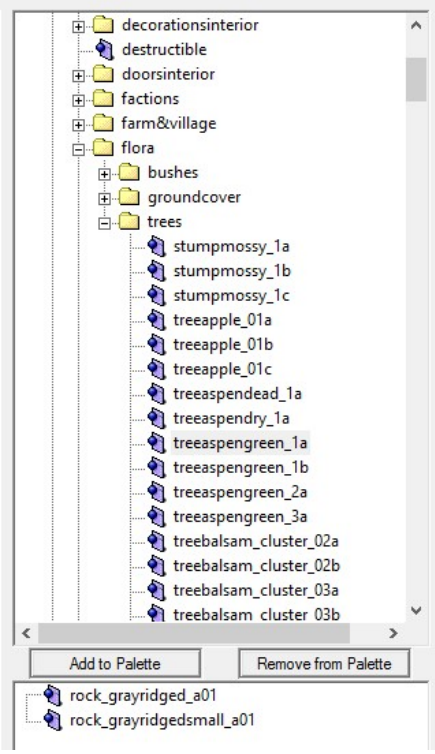


When you're finished building up the basis of your world, select one of the regions and click the Go-To button at the top.

Object Tools

Placement Tool

This tool allows you to search through the database to place objects within the world such as decorations, spawners, scripted entities and dungeon entrances. You can search this list as well as add objects to your palette for easier retrieval later.



Group Link Tool

This tool is used to link objects together, which is important for randomizing spawns, barriers and connecting Dungeon Entrances.

First you must create a group and name it, then select the type of group it is:

- Respawns, all spawn points in the world must be connected so that only one is active at any given time.
- Unified Entities, used to spawn the same object from a group of SetPiece objects.
- Proxy Patrollers, used to connect a Proxy with a set of patrol points. A proxy is grouped this way, then Linked to a group of Patrol Points.
- Unique Entities, used to randomly create only one entity out of a group. This is useful for randomizing where a Proxy or SetPiece Spawns
- Inverse Unique Entities, used to randomly create all but one of the entities in a group. This is useful for randomizing barriers such that only one way through is open.
- Any Entity, used to group together randomized Rifts. Only one will appear in the world.
- Decorations, used to group Decorations together. This is seldom used with the SetPiece system, which uses Unique Entities.
- Patrol Points, used to group Patrol Points for a Proxy to follow.
- Monsters and Weapons, this is an old setting and is no longer used with Proxies and SetPieces.
- Patrollers, used similarly to Proxy Patrollers but for individual entities such as guards.
- Wander Points, used to group Wander Points for NPCs to follow. As we keep NPCs static, this is unused.
- NPC wanderers, used similarly to Proxy Patrollers but for individual NPCs.
- Riftgates, used to link all of the Riftgates in the world so they become a network for fast travel. If a riftgate is not in this group, it will not be remembered by the player's character.
- Dungeon Entrances, used to link transition doors together. This is important for Dungeon Entrances and transitions between dungeon floors.
- Devotion Shrines, used to link all Devotion Shrines in the world so they become a network. If a shrine is not in this group, it will not be properly stored by the player's character.

Link Mode is used to connect dungeon entrances when one or both sides are randomized (ex. Randomized Rift entrance). It is important that the link is made in both directions or the entrance will not work. To link two groups together, you toggle Link Mode and click on the black Dot that indicates the center point of the group you are linking to.

Terrain Tools

The following tools all can be modified by holding Shift or Alt. Generally, holding Shift will alternate its functionality whereas Alt will prevent the terrain tool from displacing already placed assets.

Raise/Lower Tool

Use this tool to raise or lower the terrain.

Noise Tool

Use this tool to create randomized bumps in the terrain.

Smooth Tool

Use this tool to even out the variation between terrain of differing heights.

Plateau Tool

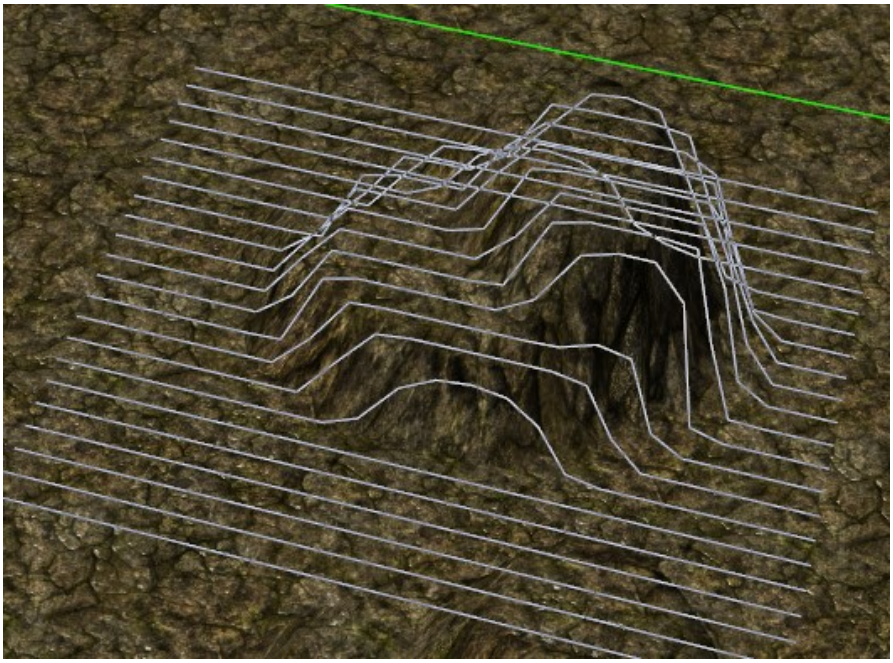
Use this tool to set a section of terrain to a fixed height.

Raise Plateau Tool

Use this tool to raise the terrain by a fixed amount. The relative difference between terrain remains the same.

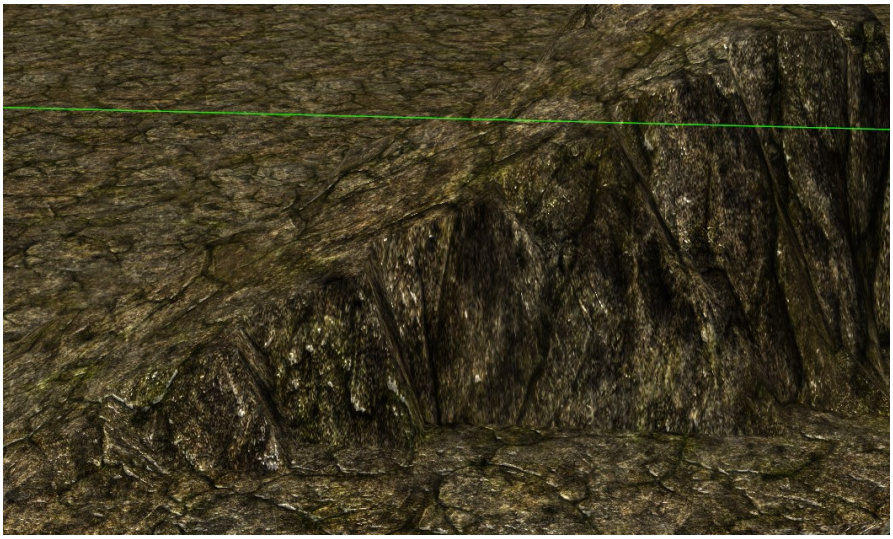
Copy Terrain Tool

This tool functions similarly to copying and pasting in a text document. CTRL+C copies an area, CTRL+V pastes it. Left-clicking also pastes your last selection.



Ramp Tool

Use this tool to create ramps from the starting point to the end point you click and hold to.



Brush Tool

Use this tool to apply terrain textures. You can also set it to apply onto paintable decorations, such as slabs of stone or bridges.

No Pathing Tool

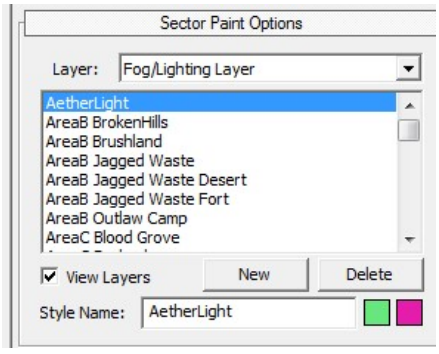
Use this tool to paint over areas you do not want players or monsters to go. Areas painted this way will be omitted when you generate pathing for your level. This only functions on terrain. Objects that generate a path mesh will override this if they are over the painted area.

Sector Paint Tool

Use this tool to apply various environmental settings, such as Lighting and Level Limits.

Fog/Lighting Layer

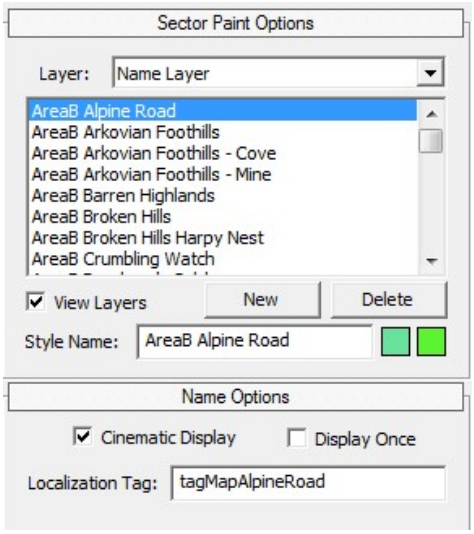
The Fog/Lighting Layer is used to control the ambience of the region. For Fog, you can set the visible distance, density, height and color. For Lighting, you can set the Light Color, the Ambient color for the sky and the Ambient color for the ground. Example, the atmosphere turns to an Aether green when entering Aetherfire. That visual is set in this layer.



Name Layer

The Name Layer determines what name appears whenever the player enters the area (ex. Devil's Crossing appears on screen when enering Devil's Crossing).

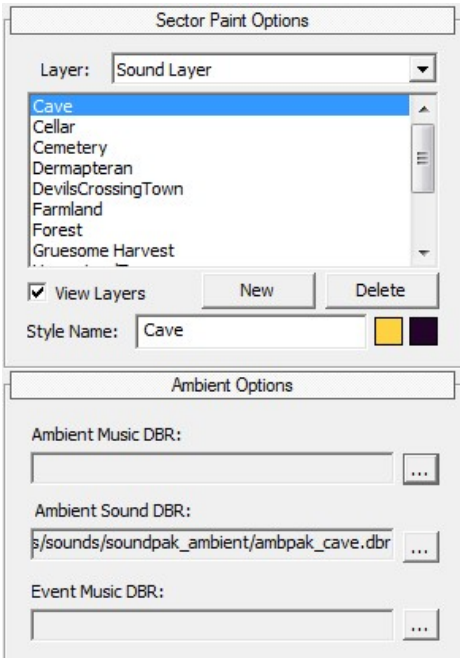
- Localization Tag – the tag reference to what players see.
- Cinematic Display – checked so the name appears in the center of the screen when first entering the area.
- Display Once – unchecked, otherwise the name only appears the first time.



Sound Layer

The Sound Layer is used to set the music and ambient sounds for an area. The sound layer references DBRs.

- Ambient Music – what music is played in this area.
- Ambient Sound – what ambient sounds randomly play in this area.
- Event Music – music triggered by a special event.

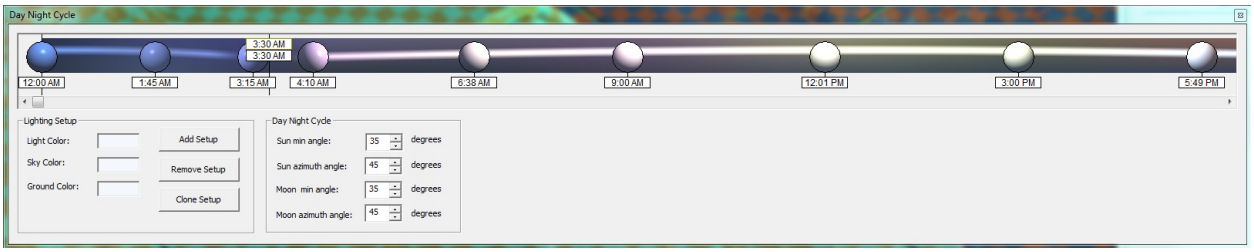


Boss Layer

The Boss Layer is used to control where special Boss Music plays. It has no settings of its own and must be associated with a Boss Monster that has custom Music assigned to it.

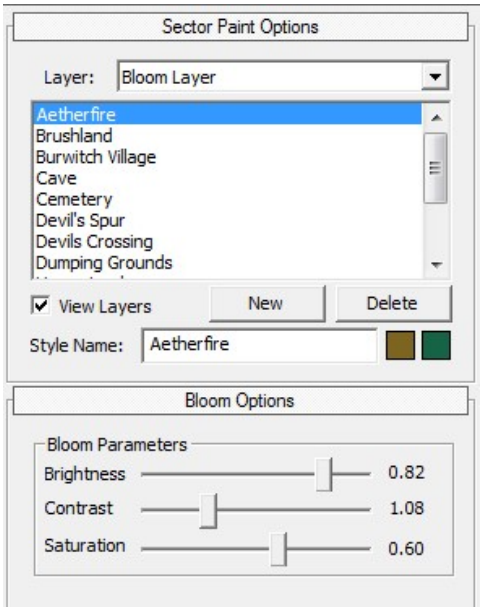
Day/Night Cycle Layer

The Day/Night Cycle Layer controls the atmospheric effects simulating sunrise/sundown.



Bloom Layer

The Bloom Layer is an atmospherics layer used to adjust the Brightness, Contrast and Saturation for the bright areas in a scene. It works similarly to HDR.

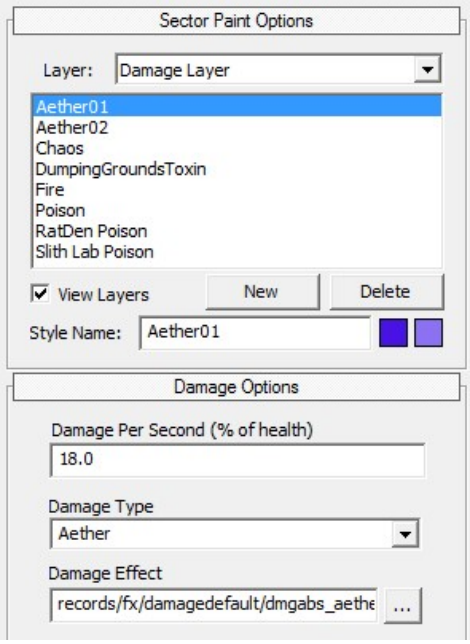


Climate Layer

The Climate Layer is used to control weather effects such as dense fog, rain, wind, etc. The Climate layer references a Climate DBR, which is where all of the settings are stored.

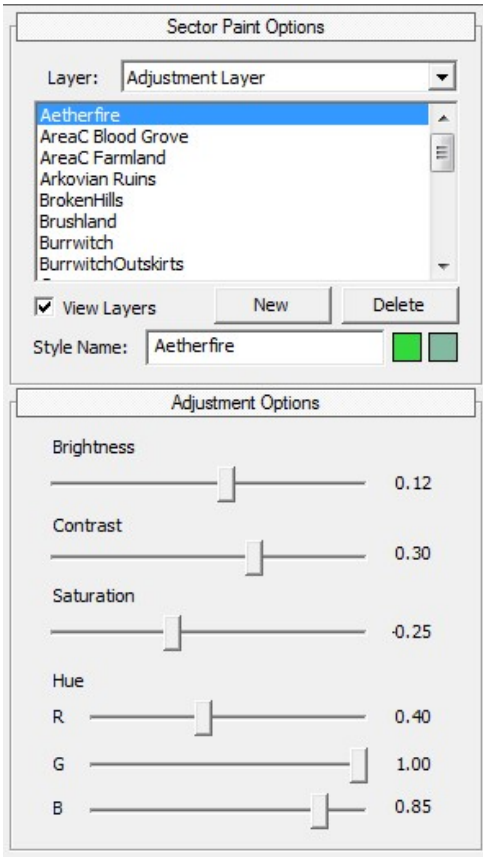
Damage Layer

The Damage Layer is used to deal damage to the player when they enter an environmental hazard. You set the type of damage, the visual effect (generally should be the one associated with the damage type) and the % health that is lost per second of standing within the hazard.



Adjustment Layer

The Adjustment Layer is another atmospheric layer which controls the Brightness, Contrast, Saturation and Hue of the entire scene. Unlike the Bloom Layer, it also impacts dark areas.



Level Limit Layer

The Level Limit Layer controls what level monsters can spawn in the area. The Minimum level and Maximum level are used whenever the player's level falls outside of that range, otherwise the player's level is used. The Offset is added to the player's level so that monsters always end up a few levels above the player (unless the player's level is above Maximum level). The Offset is only used in challenge areas.

Riftgate Disable Layer

The Riftgate Disable Layer is used to prevent players from being able to open a Personal Riftgate while within the area. It is used exclusive for very challenging areas or for Faction-Only hideouts.

View Distance Layer

This layer is used to control how far away objects are loaded, to improve performance in areas where you do not necessarily see more than a few dozen meters out. A really flat area could be set to 80m, while a really tall mountain may need 300.

PVP Layer:

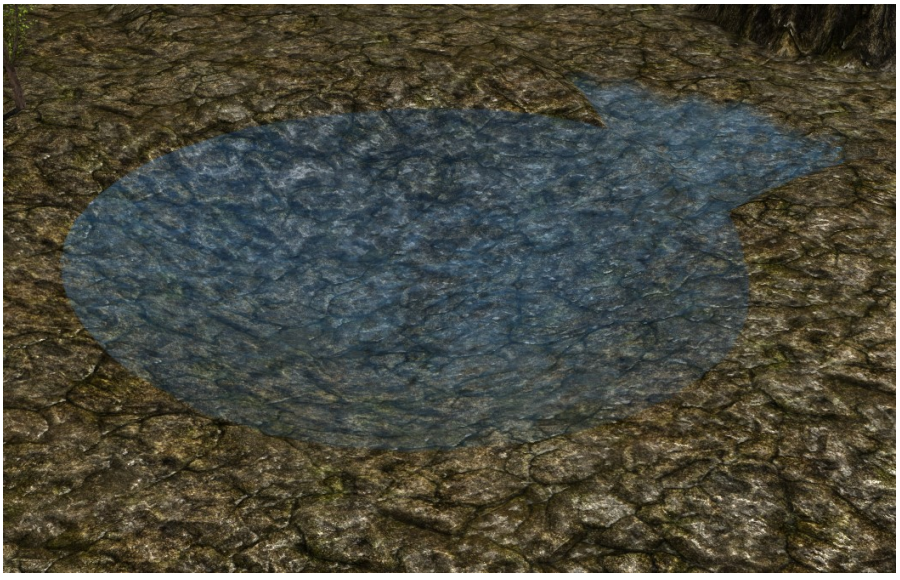
The area painted over with this layer disables Player versus Player hostility. Used to create safe zones.

Other Tools

The following tools do not interact with the terrain or server special functions.

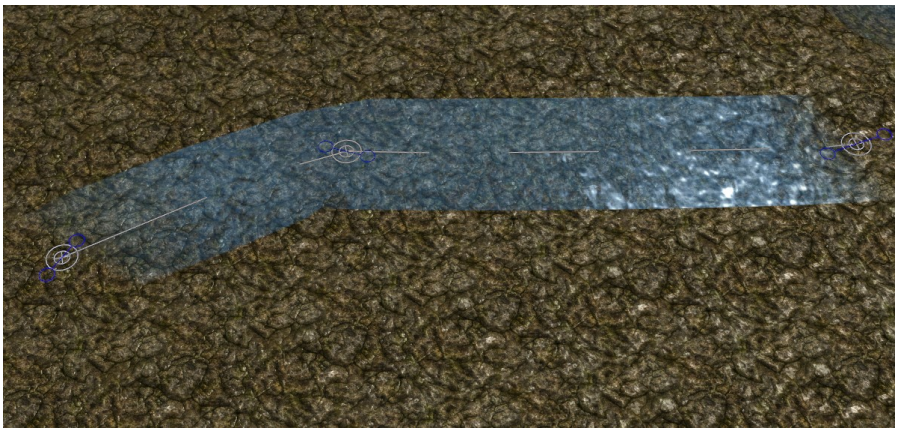
Water Tool

Use this tool to create bodies of water.



River Tool

Use this tool to place river points, which generate a flowing river. Hold Shift to place a point. Press Backspace to delete the last placed point.



Wave Tool

Use this tool to place wave points, which generate waves between them. Hold Shift to place a point. Press Backspace to delete the last placed point.

Zoom Tool

Use this tool to click and drag over an area and zoom in on it.

Undo/Redo

Use this tool to undo/redo the last Terrain or Asset action you performed. Does not interact with the Water/River/Wave tools.

Default Zoom

Click this button to reset the camera zoom to the default distance for players. Useful for seeing what the player would see without adjusting their zoom. One to two clicks out will generally show you what a player playing at 1920x1080 will see.

Default Camera Angle

Click this button to reset the camera angle to the default for players. Useful for seeing what the player would see without rotating the camera.

Updating Pathing and the Map

Once you've finished building up your level, you will need to generate pathing and a map. To do, return to Layout Mode then select all of your regions that you wish to update. Under the Build menu, select Rebuild Selected Pathing. Next, return to the Build menu and select Rebuild Selected Map.

To see if your path mesh is good and does not create any weird dead spots or tight passages, you can return to Editor Mode. Under the View menu, select Path Mesh preview.

Placing the Spawn and Respawn Points

A player will not be able to spawn in your level unless you place a spawn point. You should only have one spawn point in your level. This is where the player will begin their journey in your world.

You can also place Respawn Points, which is where the player will return if their character dies. If you use multiple Respawn Points, such as for multiple safe zones the player can reach, they will need to be grouped using the Group Link tool.

To place these points, use the Placement Tool. The Default Spawn and Respawn points are in Records -> Controllers -> Player

Finishing Up

When you are finished with your world, Save and Close the editor. Return to the Asset Manager and Build your changes.

Enjoy your level!

Tutorial 03: Quest Editor

The Grim Dawn quest tool was built to be flexible, allowing for branching resolutions and script-triggered completion. This flexibility does mean that special attention must be paid to the implementation in order to avoid errors and potential exploits, especially when scripting is involved.

Care also has to be taken to ensure that any given quest choice or incompletion of the quest does not create future dependencies or conflicts. This becomes all the more important for Multiplayer as you have to assume players can be everywhere at once.

Creating a New Quest

When you create a new quest within the Quest Editor, the first quest task is automatically created for you, but all of the important fields are left blank.

Quest Log Sub-section Name

All quests with the same text entered here will be sorted under the same sub-list in the Quest Log.

Quest Name

If you right-click this line, you can copy the Quest ID. This is important for scripting, so copy it for later reference with a “0x” added to the front (example quest ID: 6E48AA00, to use this for a script it has to be 0x6E48AA00).

Note that if you copy an existing quest, its Quest ID will also be the same, which will create issues. All quest IDs must be unique. To generate a new Quest ID, go to Quest -> Regenerate Quest ID.

Task Name

This is for internal reference only and does not appear in-game, but it is helpful to label this in a way you will recognize (ex. Kill Reanimator or Return to Bourbon) as you will see it again when connecting Quest Tasks or Conversations. If you right-click this line, you can copy the Task ID. This is important for scripting, so copy it for later reference with a “0x” added to the front (example task ID: 6483C880, to use this for a script it has to be 0x6483C880)

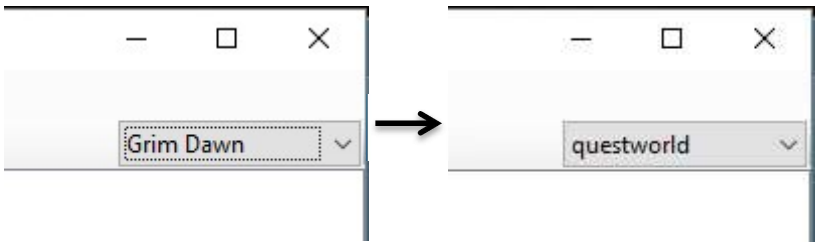
Task Description
The quest description players see in the quest log. This description is not cumulative, so when a new Task begins within a quest, it overwrites the description in the quest log.

OnAccept
The actions that are triggered when this Task begins. These can be restricted to only trigger based on given conditions.

Objectives
The objectives that must be completed for this Task. Each objective has a display text and a condition that must be met. These conditions can also trigger actions.

OnComplete
The actions that are triggered when this Task is marked as complete, which will occur automatically when all objectives are completed or via a call from a Script/Dialogue. This is where quest rewards go for the final task.

Quest Conditions and Actions for Mods
By default, the Quest Editor only detects quest lists and Lua scripts from within the base game. In order to assign quests and functions from mods into conditions/actions, you must change the instance of the Quest Editor to the respective mod. To do so, select the mod you wish to work with from the drop-down on the top-right of the editor.



By default, this is set to Grim Dawn (the base game). The Quest Editor will remember what mod it was last set to for ease of use.

Quests Tasks
Quests are sorted into Tasks. Each Task is a collection of objectives, conditions, and actions. A quest can contain multiple Tasks, but the player should never be on more than one Task within a quest at any given time. You can think of Tasks as chapters within a short story.

You can add additional Tasks by double-clicking the Quest Name. Tasks can be chained together with a series of “CompleteQuestTask” and “BeginQuestTask” actions, or via an NPC dialogue that performs those same Actions.

Tokens:

Tokens are special text tags that are stored in the player's save data. These tags can then be referenced within quests, scripts and NPC dialogue. They are essentially true/false variables in that you test for whether a player has a specific tag or not. They can be granted to a player via dialogue, quests, and scripts. Items, monsters, and trigger volumes can bestow tokens by calling a script.

Example 1:

The player kills a quest monster. The monster grants a token confirming that the player has killed it even though he was not on the quest. If the player acquires the quest in the future, he can claim credit for the kill without having to hunt the monster down again.

Example 2:

The player finds a key. Upon picking it up, the key grants a token which grants access to a special dungeon.

Completing a Quest

For the final task of a quest (this could be multiple tasks if there are choices involved), you will want to do several things:

- Run the CompleteQuest Action on the quest
- Remove or replace any Tokens the player no longer needs
- Give out rewards based on the game difficulty

Main Quest versus Side Quest

If a quest is part of your Main Quest line, as opposed to a side quest, you can designate it as such by right-clicking on the Quest Name field and checking "Main Quest".

Repeatable Quest

If you wish for a quest to be repeatable by the player, you can mark it as Repeatable by right-clicking on the Quest Name field and checking “Repeatable”. This will allow the player to accept the quest again and again, even in the same game session.

Special Cases for Multiplayer

When dealing with multiplayer, there are two more special tags you can apply which can help resolve unusual circumstances creating by having more than one player in the game.

Don’t Propagate

If you have a task that you wish for each player to complete individually (ex. turning in a special item), you can mark the task to not automatically propagate completion to all players that are also on the same task. To do so, right-click the task and check Don’t Propagate.

Blocker Task

Normally, if a quest is already completed by the host, players who join can no longer complete that quest in that game session. This is done to prevent conflicts in dialogue or the executing of scripts that should not be run multiple times (or ones that already ran and create paradoxes in the joining player’s story progression).

However, this automatic system does not apply for when the host and a joining player are on the same quest, but the host is a task or more ahead. Sometimes this works out fine (ex. Requiring a player to Find A Location can be automatically completed for all players present if the host already found this location), but it can cause a problem if the host has just killed a boss that a joining player still needs. The joining player would then have a quest they cannot complete in that session.

To prevent such scenarios, you can right-click on a task and check Blocker Task, so that a player joining the game on the same quest as the host but after the quest target has been dealt with will have the quest properly Blocked.

Quest Markers

To aid players in find quest targets, you can mark them. Whether you are marking a monster, item, or dungeon entrance, you need to edit the DBR section called Quest Marker. There you can set the quest file, the Task ID and the markerRange (distance at which it appears on the minimap).

You can also create an Area of Interest, which is a DBR entry that allows you to place an invisible dot in the game world. These markers are also good for marking dungeon entrances on the local map. If a marker is only for a quest, set questOnly to true.

Finishing Up

When you are finished with your quest, Save and Close the Quest Editor. Return to the Asset Manager to create an asset for it, as quests require one before they can function in the game.

Before you can go off and send players on your quest though, you have to add the quest to your world by using the World Editor. With your world loaded, go to the Quest Menu -> Select Files. Add your new quest to this list and save on exit.

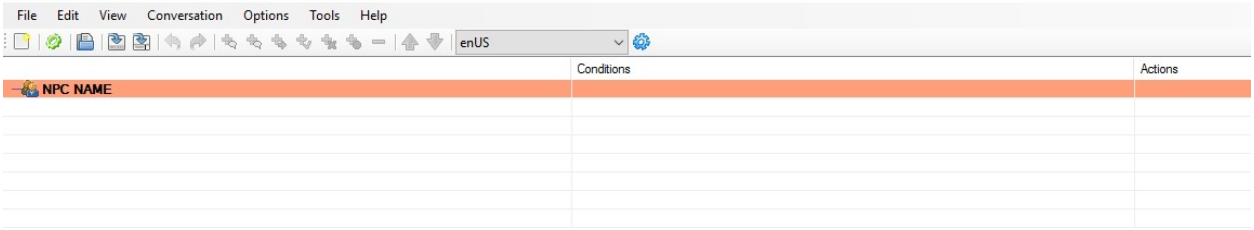
Tutorial 04: Conversation Editor

The Grim Dawn conversation tool is versatile, capable of handling complex multi-ending NPC conversations and executing scripts. Because of this there are some important considerations that must be kept in mind when creating NPC conversation logic.

Because of the possibility of choice-and-consequence in Grim Dawn's questing, only one player can talk to an NPC at a time. However, as players in a multiplayer session can be in many places at once, care must be taken when creating NPCs that impact each other's dialogues.

Creating a New Conversation

When you create a new conversation within the Conversation Editor, the only field available will be the Name field that is displayed in the NPC's dialogue window. You should have this match the name of the NPC that is talking.

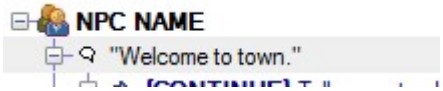


At this point you need to add Speech Nodes, which in turn will have their own Player Dialogue Nodes branching off. Using these you can create an intricate dialogue web with many resolutions and bits of information. An NPC can have as many things to say as you desire.

For all of the types of dialogue nodes, you can execute both Conditions, which determine if the player can see the speech or dialogue option, and Actions, which execute various quest and script events.

Speech Node

When you double-click the NPC Name field, a new Speech Node will be created. Speech Nodes are the text that the NPC will display to the player.



Speech Nodes are processed from top to bottom. What this means is that if a Node at the top is valid, it will be used even if another valid Node exists underneath it. This is important for managing quest logic for multi-ending or multi-quest NPCs. Because of this, you need to carefully set Conditions for each Speech Node.

Typically, the bottom-most Node has no condition and serves as a catch-all, or starting, Dialogue. This dialogue can then lead to others, such as a quest offer or quest turn-in, which would have Conditions the player has to meet (example Player Is on Quest Task A and Has Item).

Continue Node

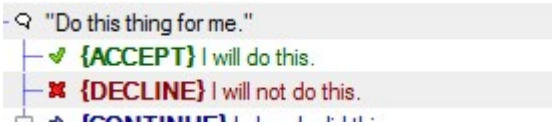
A Continue Node is a player response that leads to another Speech Node. Use this when the player can choose where a conversation goes next or to ask a question.

End Conversation Node

An End Conversation Node does exactly as it implies: it allows the player to end the conversation. It can still execute Actions though, which is useful for turning in quests or making an NPC swap into a hostile foe.

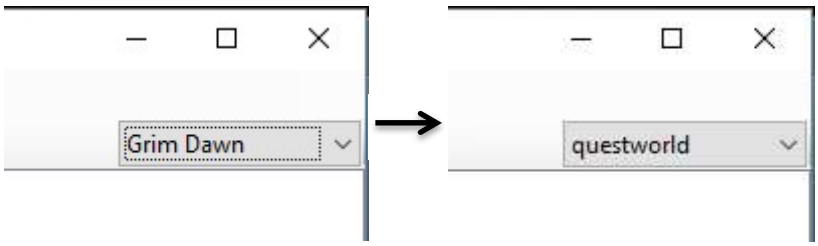
Accept/Decline Node

An Accept Node is a special type of End Conversation Node used when you want to use a Checkmark and X choice for the player, such as when offering a quest. Generally, an Accept node will execute Actions whereas a Decline node simply ends the conversation.



Conversation Conditions and Actions for Mods

By default, the Conversation Editor only detects quest lists and Lua scripts from within the base game. In order to assign quests and functions from mods into conditions/actions, you must change the instance of the Conversation Editor to the respective mod. To do so, select the mod you wish to work with from the drop-down on the top-right of the editor.



By default, this is set to Grim Dawn (the base game). The Conversation Editor will remember what mod it was last set to for ease of use.

Granting a Quest in Multiplayer

When a player accepts a quest via a Conversation, this does not automatically propagate to all players in a session, as those players did not necessarily go through the same Conversation logic to reach the same quest offer and may actually not be eligible.

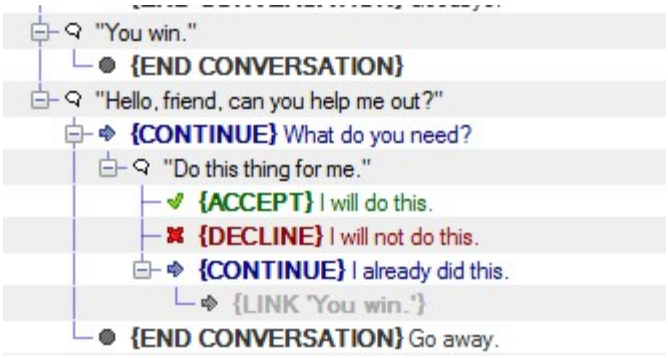
To handle this, we execute a global quest offer via Lua Scripting. Lua Scripting for quests is covered in Tutorial 07.

```
(BeginQuestTask sq_finalsalute.qst, [1] Find Insignias and return), (LuaScript QuestGlobalEvent("finalSaluteBeginQuest"))
```

If you wish to make a gameplay scenario for Multiplayer, we highly recommend studying the Granting Multiplayer Quests section of the Lua Scripting tutorial.

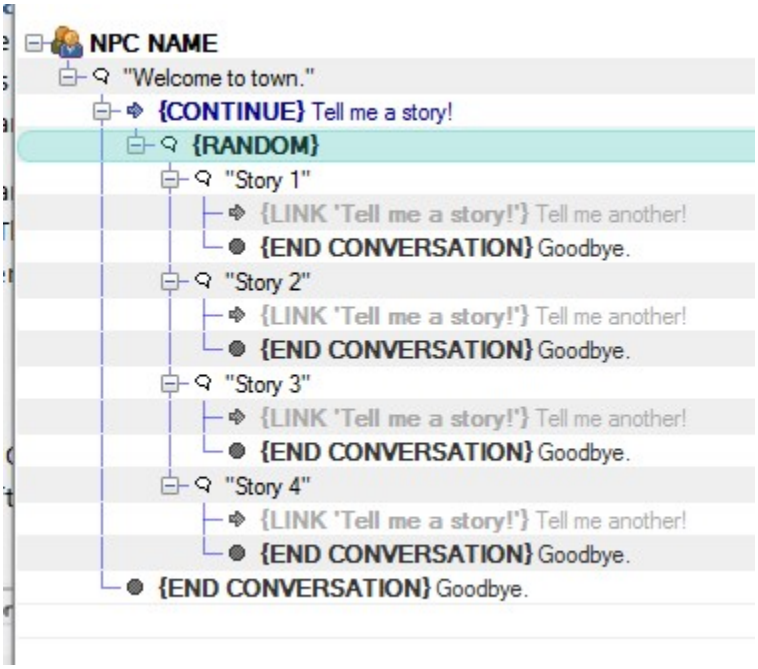
Linking Dialogues

When you have a dialogue that logically leads to another existing one, you can link the two dialogues together without rewriting the same thing several times. To do so, simply Copy (CTRL+C) a dialogue node (can be any type except End Conversation), then select the dialogue you want leading to this same node and Paste as Link (CTRL+L). The dialogue you are linking has to logically connect, so you cannot have a Speech Node connect to another Speech Node, but you can have a Speech Node linked to several Continue Nodes.



Random Speech

A random speech node is a collection of Speech Nodes that will randomly be selected. Use this to have an NPC tell the player a random quip each time they are visited.



Assigning Voiceovers

If you decide to include your own voice-overs for NPCs, you will need to assign the recordings to each Speech Node. To do so, right-click the desired Speech Node and select the appropriate .wav Asset from the list.

Finishing Up

When you are finished with your conversation, Save and Close the Conversation Editor. Return to the Asset Manager to create an asset for it, as conversations require one before they can function in the game.

To assign a Conversation to an NPC, open its DBR and go to the Conversation section and assign the appropriate Conversation Asset.

Tutorial 05: Creating a Monster

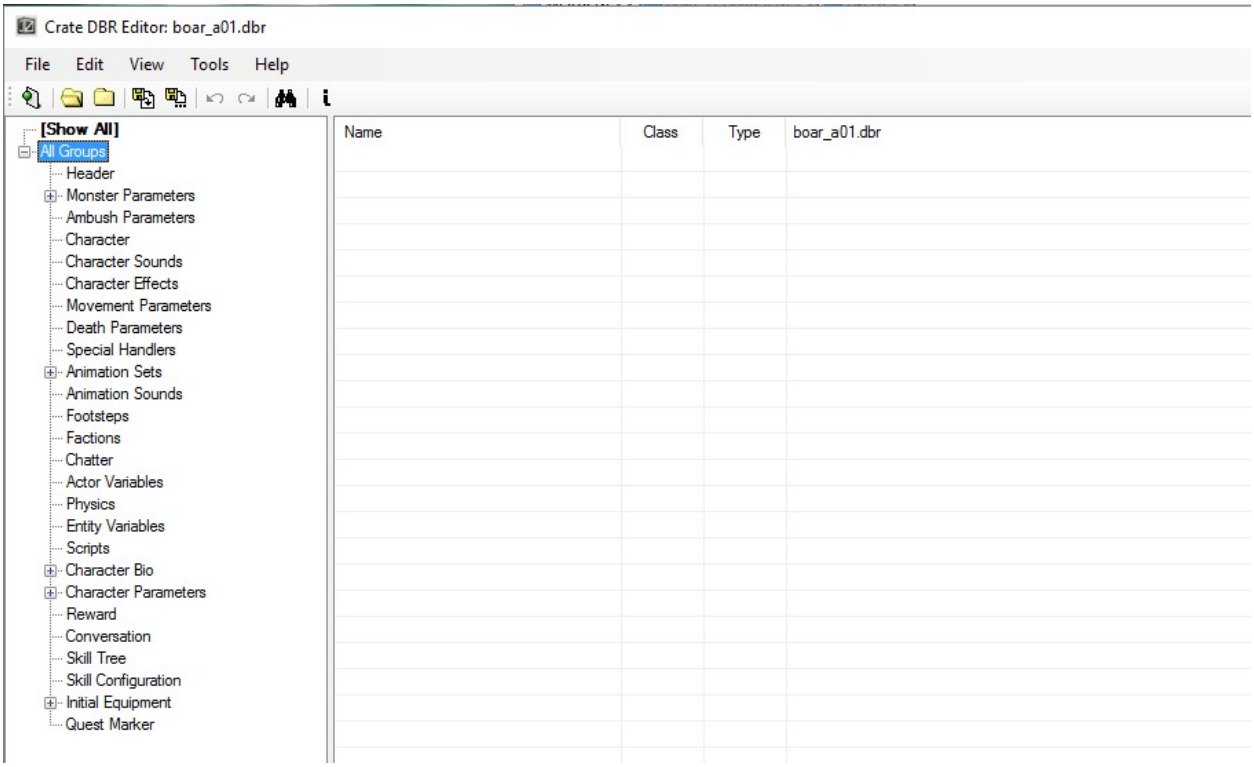
Monsters are created and modified using Grim Dawn's Database Record (DBR) Editor. There are many parts that go into making a monster, which are broken down in this tutorial.

1) Dig up appropriate body parts
2) Combine them in Primordial broth
3) Stir
4) Stir some more
5) Name the monster
6) Run faster than the guy next to you

Monster Record

The first step to creating a monster is to create its Database Record (DBR). The Monster Record references other records and assets, such as sounds and animations. Generally it is better to take an existing DBR and modify it as there are many data fields that must be entered which will not be covered in this guide.

For the purpose of this tutorial, we will assume that you are not making new models, animations or sound files but instead will be basing your monster on an existing archetype.



Monster Parameters

Monster Controller

The Monster Controller is a separate DBR that sets various parameters for the monster such as when to flee, when to assist, vision range, hate generation (threat), and more. For the most part, you will be using existing Controllers.

Monster Classification

The monster classification determines the color of the monster's name in the Target Info section of the HUD as well as the multipliers the monster uses to determine the quality of its loot.

- Common: used for fodder monsters that pose minimal challenge individually
- Champion: used for mid-tier monsters that pose some challenge individually but throw in additional mechanics in a group fight and should be prioritized.
- Hero: used for hero monsters with custom skill sets. They are marked by a hero star above their head.
- Quest: used for boss and mini-boss monsters with custom skill sets.
- Boss: used for Nemesis bosses.

Per Party Member Drop Item Name/Chance

This parameter is used to set what item this monster drops per player in a multiplayer session. You can also set a chance if you do not wish for it to drop 100% of the time. Useful for quest drops and lore notes.

Drop Items

A true/false check for whether the monster can drop what it has equipped. Generally set to False for summons and true for everything else.

Give XP

A true/false check for whether the monster grants XP when killed. Generally set to False for summons and true for everything else.

Health Gain on Kill Percent/Difficulty

The % Health the monster regains if it kills a player. The Difficulty determines at what difficulty this health regains activates (0 = Normal, 1 = Veteran, 2 = Elite, 3 = Ultimate). For all bosses, the health regain starts on Veteran. Nemesis bosses regain health regardless of difficulty.

Character

Char Level

The Char Level is an equation for the monster's level relative to the player's. This is only used if the monster is placed directly into the world and not spawned via Proxy. The Min/Max Level parameters are caps on the formula. (example formula: charLevel*1+3)

Experience Points
Bonus experience for killing this monster, generally only set for heroes and above.

Character Gender Profile
The Gender Profile is used only for Human-based monsters as it determines the equipment gender overrides (if set incorrectly you could end up with A female character using a male torso).

Character Racial Profile
The Racial Profile is used to tag the monster with various Cairn races for the purposes of Bonus Damage. A monster may have more than one profile at a time (Example: Human Cultists are both Human and Bloodsworn). The default game races are as follows:

- Race001 = Undead
- Race002 = Beastkin
- Race003 = Aetherial
- Race004 = Chthonic
- Race005 = Aether Corruption
- Race006 = Bloodsworn
- Race007 = Eldritch
- Race008 = Insectoid
- Race009 = Human
- Race010 = Construct
- Race011 = Riftspawn
- Race012 = Beast
- Race013 = Magical
- Race014 = Celestial
- Race015 = Arachnid

Default Equipment
The default equipment is used to give the monster items to visually change its appearance without that equipment having a chance to drop. This is mostly relevant for Human-based monsters that wear player equipment visually, but is also relevant for tribal monsters where the champions might wear armor (that the player cannot wear) while the fodder monsters do not.

Weapon Scale
Weapon Scale is used to scale up the model size of any weapons the monster uses, useful for large monsters that use player gear. When the weapons drop, they are reduced back down to the original scale.

Number Attack/Defense Slots
Attack Slots are important for very large monsters so that pets and other entities can attack them at the same time. The Default of 4 is sufficient in most cases, but a particularly large

monster may need more if it is difficult for entities to reach it from multiple sides (example: the Loghorrean uses 26).

Start Visible

A true/false check whether the monster initially spawns visible. This is important for monsters that are ambushing from above. These monsters must be set to appear visible in the spawn animation or else they will be permanently invisible.

Distress Call

A true/false check on whether the monster calls for help. If set to true, the range determines how far the distress call reaches, the group determines the group of monsters that is called (all monsters with the same group set here and in the controller will respond), the call time is the delay between additional distress calls and the max calls is the maximum number of times the monster will call for help.

Invincible

If set to true, this monster cannot be killed. This is useful for environment traps.

Hidden From Combat

If set to true, the monster will not cause other non-player entities to attack it.

Causes Anger / Anger Multiplier

A true/false check on whether the monster generates threat with other non-player entities. The multiplier can be used to reduce its threat generated without completely disabling it.

Upper/Lower Health Display Percentage

Used for bosses with multiple phases. This way you can keep one continuous health progress bar across all phases. The Upper is the starting % of health and Lower is the ending % of Health when the next phase would begin. (Example: Phase 1 would be 100 for Upper and 50 for Lower and Phase 2 would be 50 for Upper and 0 for Lower in a two phase encounter).

Hit Threshold

The % of health the monster must lose in one hit before it is staggered by the blow. Bosses generally have this set to a really high value so that they are immune to this mechanic.

Targetable

A true/false check for whether a player can target the monster with attacks and display its info on the HUD.

Factions

The monster's faction is set here, which is set up in a separate DBR. This determines what faction the player gains/loses faction status with when killing the monster. It also determines what factions the monster considers a foe, which can result in monster in-fighting.

Actor Variables

This section is mainly used to change the monster's physical appearance, but the Description field is used to assign a tag for the monster's name.

Scripts

Under "Scripts", you would assign Lua hooks for various trigger events. The most common one is onDie, for when the monster is killed. This is often used to give out kill-credit tokens and open doors.

Character Bio

This is where the monsters BIO record is assigned, where its health, energy and other base attributes are set.

BIO Record

The BIO Record controls the monster's Physique (Strength), Cunning (Dexterity), Spirit (Intelligence), Health (Life), Energy (Mana), Regeneration, Offensive Ability and Defensive Ability. Each of these fields is an equation (example: (charLevel*5)+20) so that the monster's attributes scale with its level.

Name	Class	Type	bio_boar_01.dbr
ActorName	Variable	String	
Class	Static	String	
FileDescription	Variable	String	
characterStrength	Variable	Equation	(charLevel*5.4)+15
characterDexterity	Variable	Equation	(charLevel*6.5)+10
characterIntelligence	Variable	Equation	(charLevel*3)+20
characterLife	Variable	Equation	((charLevel*6)^1.28)+20
characterMana	Variable	Equation	((charLevel*6)^1.22)+60
characterLifeRegen	Variable	Equation	(((charLevel/15+1) + lifeRegen) * (1 + lifeRegenMod/100))*elapsedTime
characterManaRegen	Variable	Equation	(((charLevel/10+3) + manaRegen) * (1 + manaRegenMod/100))*elapsedTime
characterOffensiveAbility	Variable	Equation	(charLevel*6)+30
characterDefensiveAbility	Variable	Equation	(charLevel*3)+20

Natural Resistance

This sub-section is where you set the monster's baseline resists to various damage types and forms of crowd control.

Character Parameters

The only sub-section relevant here is the Character Speed, as the others are handled via the BIO Record. Here you set the monster's default speed modifiers (Run, Attack Speed, SpellCast Speed). The Run Speed Jitter is used to apply additional variation (0-100) so that packs of the same monster will not all run at the same speed and form more of a spread-out pack rather than a clump of enemies.

Reward

This section is used only to apply a money generator to the monster and the chance of dispensing it. Additional loot rewards are set in the Initial Equipment section.

Initial Equipment

This section determines the equipment the monster is wearing and actively benefiting from. This means that any bonuses the gear has will be applied to the monster. It also means that the equipment will drop as part of the WYSIWYG Loot System.

The only exception to this is the Misc sub-section, where you can assign loot that is not equipped, but can still drop from the monster when killed (example: components, Constitution Rations, rare crafting materials, etc.). This can reference specific items or Loot Tables (Master Table or Dynamic Table, but not Level Table)

Name	Class	Type	boar_a01.dbr
chanceToEquipMisc1	Variable	Real	4
chanceToEquipMisc1Item1	Variable	Int	5
chanceToEquipMisc1Item2	Variable	Int	55
chanceToEquipMisc1Item3	Variable	Int	40
chanceToEquipMisc1Item4	Variable	Int	0
chanceToEquipMisc1Item5	Variable	Int	0
chanceToEquipMisc1Item6	Variable	Int	0
lootMisc1Item1	Array	DBR	records/items/loottables/mastertables/mt_comp_rare_a01.dbr
lootMisc1Item2	Array	DBR	records/items/loottables/mastertables/mt_comp_beastfur_a01.dbr
lootMisc1Item3	Array	DBR	records/items/materia/compa_bristlyfur.dbr
lootMisc1Item4	Array	DBR	
lootMisc1Item5	Array	DBR	
lootMisc1Item6	Array	DBR	

Skill Tree

This is where all of the monster's skills are assigned. The skill level can be a fixed value, or a formula that allows the skill to scale with the monster's level.

Name	Class	Type	boar_a01.dbr
skillName1	Variable	DBR	records/skills/nonplayerskills/passive/damage_totaladjuster.dbr
skillLevel1	Variable	Equation	(charLevel/30)+1
skillName2	Variable	DBR	records/skills/nonplayerskills/passive/damagebase_physical01.dbr
skillLevel2	Variable	Equation	charLevel*1
skillName3	Variable	DBR	records/skills/nonplayerskills/passive/armorbase02.dbr
skillLevel3	Variable	Equation	charLevel*1
skillName4	Variable	DBR	records/skills/nonplayerskills/passive/passiveproperties_boar.dbr
skillLevel4	Variable	Equation	charLevel/4+1
skillName5	Variable	DBR	records/skills/nonplayerskills/attackcharge/genericphysical_charge.dbr
skillLevel5	Variable	Equation	2

Skill Configuration

This section is where you control when and how often a monster uses its skills.

Name	Class	Type	boar_a01.dbr
attackSkillName	Variable	DBR	
buffSelfSkillName	Variable	DBR	
buffSelf2SkillName	Variable	DBR	
buffSelf3SkillName	Variable	DBR	
buffOtherSkillName	Variable	DBR	
buffOther2SkillName	Variable	DBR	
buffOther3SkillName	Variable	DBR	
nightBuffSkill	Variable	DBR	
healSkillName	Variable	DBR	
healSkillDelay	Variable	Real	
berserkSkillName	Variable	DBR	
dyingSkillName	Variable	DBR	
initialSkillName	Variable	DBR	
specialAttackSkillName	Variable	DBR	records/skills/nonplayerskills/attackcharge/genericphysical_charge.dbr
specialAttackTimeout	Variable	Real	6
specialAttackDelay	Variable	Real	8
specialAttackChance	Variable	Real	50
specialAttackRange	PickList	String	LongRange
specialAttack2SkillName	Variable	DBR	
specialAttack2Timeout	Variable	Real	
specialAttack2Delay	Variable	Real	
specialAttack2Chance	Variable	Real	
specialAttack2Range	PickList	String	
specialAttack3SkillName	Variable	DBR	
specialAttack3Timeout	Variable	Real	
specialAttack3Delay	Variable	Real	
specialAttack3Chance	Variable	Real	
specialAttack3Range	PickList	String	
specialAttack4SkillName	Variable	DBR	
specialAttack4Timeout	Variable	Real	
specialAttack4Delay	Variable	Real	
specialAttack4Chance	Variable	Real	
specialAttack4Range	PickList	String	
specialAttack5SkillName	Variable	DBR	
specialAttack5Timeout	Variable	Real	
specialAttack5Delay	Variable	Real	
specialAttack5Chance	Variable	Real	
specialAttack5Range	PickList	String	
shortRangeMin	Variable	Real	0
shortRangeMax	Variable	Real	4
mediumRangeMin	Variable	Real	4
mediumRangeMax	Variable	Real	15
longRangeMin	Variable	Real	12
longRangeMax	Variable	Real	20
chainInitialSkill	Variable	DBR	
chainNextSkill	Variable	DBR	
chainBehavior	PickList	String	

- **AttackSkill** overrides the monster's default auto-attacks with this skill. This is useful for monsters that throw spell projectiles as their default attack.

- **buffSelfSkillName** tells the monster to use this skill to buff itself, whether there are allies nearby or not. The skill must have a cooldown set in the skill itself, or else the monster will spam it and get stuck.
- **buffOtherSkillName** tells the monster to use this skill only to buff allies. It will not use this skill unless allies are around to be buffed, even if it does not have the buff on itself. The cooldown for this skill should be set on the skill itself, or else it may get spammed.
- **nightBuffSkill** is a toggled buff activated only during the night. Used to make monsters tougher at night.
- **healSkill** tells the monster to use this skill to heal allies. It will not use it to heal itself.
- **healSkillDelay** tells the monster to wait this many seconds minimum between heals.
- **berserkSkillName** tells the monster to use this skill when it is low on health. Paired with lowHealthTriggerLevel, which is a percent entered as 0-100.
- **dyingSkillName** tells the monster to cast this skill when it dies. Useful for creating damaging fields or explosions on death.
- **initialSkillName** tells the monster to cast this skill when it spawns. This is used for monster auras.
- **specialAttack#SkillName** controls specific skills, up to 5, based on the sub-parameters
- **attackTimeout** is how many seconds into combat the monster will wait before using this skill.
- **attackDelay** is how many seconds the monster will wait between uses of this skill
- **attackChance** is the chance that the monster will use the skill whenever the Delay runs out. If this chance is not set to 100%, the monster may not use the skill and instead trigger another Delay.
- **attackRange** is the range the player must be in for the monster to consider using the skill. The options are Any, Short, Medium and Long. The values for these are set at the bottom. Usually, Short is 0-4 for melee attacks, the others vary based on need.
- **chainInitialSkill** is used to tell the monster to use a specific sequence of skills. The initialSkill is the triggering skill. The chainNextSkill is what skill it will use after the triggering skill is used. The chainBehavior tells the monster what the target is for the NextSkill. The target is usually CurrentEnemy.

Quest Marker

This is where you set what quests, if any, the monster is marked for on the map. You need to know the quest File and the task ID. If the monster is relevant for more than one task in a quest, it needs multiple entries.

The Marker Range determines how many meters away the player can be before the monster appears on the map.

Monster Animations, Appearance and Sounds

These are references for advanced users who will be creating custom assets.

Animation

Monster animations are stored in a CharAnimationTable Record. This Record is assigned to the monster under the Animation Set. This section has sub-fields where you can enter individual animations for the monster, but generally you should use an Animation Set. If you have a monster with very limited animations, such as a trap, then it makes sense to assign them directly to the monster.

Appearance

Monster appearance is dictated by several data fields in the Actor Variables section. Here you can set the Base Mesh, Shader, and Texture Overrides (Base, Bump, Spec and Glow).

Sounds

Monster sounds are a located in the following sections:

Monster Parameters

Ambient, Alert, Rally, Rampage and Flee

Character Sounds

Attacks and Stunned

Animation Sounds

Critically Hit, Death, Vox, Body Fall and Special Animation References

Footsteps

Footsteps

Spawning Monsters

Generally when you place monsters in the world, you will want to do so via Proxies. Proxies can dispense multiple enemies in an area, or even create an ambush.

Standard Proxy

A standard proxy dispenses monsters upon being loaded, based on its parameters.

Name	Class	Type	spidergiant_n.dbr
ActorName	Variable	String	
Class	Static	String	Proxy
FileDescription	Variable	String	
difficultyLimitsFile	Variable	DBR	
chanceToRun	Variable	Real	100
placementExtents	Variable	Real	10
delayedRun	Variable	Bool	FALSE
factionRequired	PickList	String	
factionGreaterThan	Variable	Bool	
factionStanding	PickList	String	
difficultyAtLeast	PickList	String	
pool1	Variable	DBR	records/proxies/pools/p_spidergiant_n.dbr
weight1	Variable	Int	100
time1	PickList	String	
pool2	Variable	DBR	
weight2	Variable	Int	
time2	PickList	String	
pool3	Variable	DBR	
weight3	Variable	Int	
time3	PickList	String	

- Chance to Run: The Chance for this proxy to dispense upon load.
- Placement Extents: The area that this proxy will dispense monsters into, a radius around its point of origin.
- Delayed Run: Used for Scripts and Devotion Shrines only.
- Faction Required: The Faction required for this Proxy to dispense.
- Faction Greater Than: True/false check for whether the player's Faction status has to be greater or less than the standing set.
- Faction Standing: The required Faction status.
- Difficulty At Least: The minimum difficulty this proxy dispenses on (Normal, Epic or Ultimate)
- Pool #: The Spawn Pool this Proxy dispenses, it can randomly choose between several Pools, if more than one is assigned.
- Weight #: The relative weight of picking this Pool #.
- Time #: The time of day this Spawn Pool can dispense during (Always, Day or Night).

Ambush Proxy

An ambush proxy only dispenses when a player comes near its alert area. It has some additional parameters that a standard proxy does not use.

Name	Class	Type	spidergiant_amb_n.dbr
ActorName	Variable	String	
Class	Static	String	ProxyAmbush
FileDescription	Variable	String	
alertArea	Variable	Real	6
minSpawnTime	Variable	Real	0.5
maxSpawnTime	Variable	Real	
minDelayTime	Variable	Real	
maxDelayTime	Variable	Real	3.5
minGroupSize	Variable	Int	3
maxGroupSize	Variable	Int	4
spawnThreshold	Variable	Int	2

- Alert Area: The trigger area around the point of origin for this proxy that the player must enter to trigger the spawn.
- Min/Max Spawn Time: The minimum/maximum time in seconds over which the proxy will dispense monsters.
- Min/Max Delay Time: The minimum/maximum delay in seconds after triggering before the proxy dispenses monsters.
- Min/Max Group Size: The minimum/maximum number of monsters the proxy will dispense at a time.
- Spawn Threshold: The number of monsters that have to be remaining before more are dispensed (if the total from the spawn pool was not already dispensed).

Spawn Pool

Spawn Pools are referenced by proxies to spawn a group of monsters. The distinction between the Regular and Champion Pools is solely for controlling their spawn counts. You can spawn a Champion monster from the Regular Pool. It makes no special distinction.

spawnMin	Variable	Int	6
spawnMax	Variable	Int	9
championMin	Variable	Int	0
championMax	Variable	Int	1
championChance	Variable	Real	10
proxyPoolEquation	Variable	DBR	records/proxies/proxypoolequation_01.dbr
ignoreGameBalance	Variable	Bool	
name1	Variable	DBR	records/creatures/enemies/spidergianta_a01.dbr
weight1	Variable	Int	30
levelVarianceEquation1	Variable	DBR	records/proxies/lv2_normal.dbr
limit1	Variable	Int	
minPlayerLevel1	Variable	Int	
maxPlayerLevel1	Variable	Int	
alwaysSpawn1	Variable	Bool	

- Spawn Min/Max: The number of monsters to spawn from the Regular Pool.
- Champion Min/Max: The number of monsters to spawn from the Champion Pool.
- Champion Chance: The chance of dispensing from the Champion Pool.
- Proxy Pool Equation: Record reference for modifying the spawn min/max values. The default proxypoolequation_01.dbr is used for all Grim Dawn spawn pools.
- Ignore Game Balance: True/false check whether to use the difficulty modifiers that increase spawn count. This is important for proxies that dispense a single entity, such as Aether Crystals, as otherwise several crystals could spawn on top of each other.
- Name #: Record reference to a monster DBR.
- Weight #: The relative weight of dispensing this particular monster.
- Level Variance Equation #: Record reference to a level formula for the monster to spawn at. This overrides the level formula set in the monster's DBR.
- Limit #: the maximum number of this monster that can spawn from this pool.
- Min/Max Player Level #: The minimum/maximum player levels at which the monster will appear. For example, heroes do not appear before level 8, so the min would be set to 8.
- Always Spawn #: True/false check for whether this monster always spawns with this pool. Should always be coupled with a Limit as otherwise every monster that the pool dispenses will be this monster.

Finishing Up

Compile your mod and you can place the monster in your level directly, or via a Proxy. You are all set!

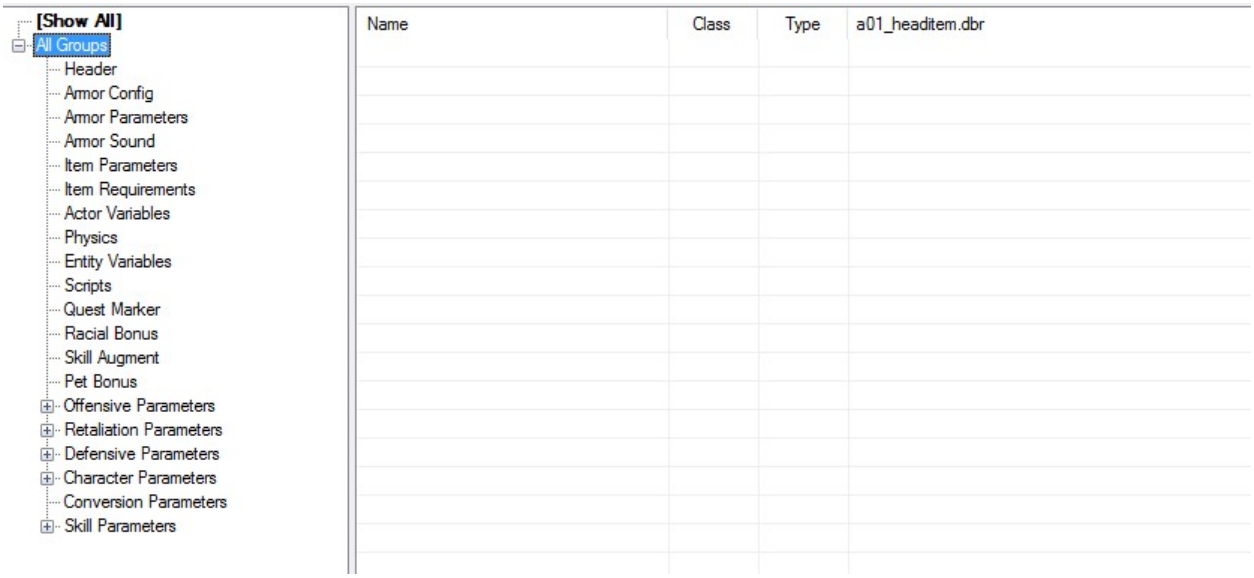
Tutorial 06: Creating Equipment

Equipment is created and modified using Grim Dawn's Database Record (DBR) Editor.

Item Record

The first step to creating an item is to create its Database Record (DBR). As items have varying gear slots, the template for this record will vary, but the data fields are mostly the same regardless of item.

For the purpose of this tutorial, we will assume that you are not making new models, textures, bitmaps and sound files for items but instead will be basing your equipment on existing assets.



Armor Config

In this section, you set the type of armor (Caster, Light or Heavy). This is mainly for display purposes as the armor requirements and protection values are set independently of this.

Weapon Projectiles

This section is only available for ranged weapons. You can assign a custom projectile as well as a chance for the weapon attacks to pierce through enemies.

Weapon Parameters

In this section, you can assign a custom weapon trail and attack effect.

Shield Config

This section is only available for shields.

Defensive Block Chance

The chance to block

Defensive Block
The damage blocked by the shield

Block Absorption
The % of damage that is absorbed by the shield, whereas the rest always goes through (the same way Armor Absorption works). This is set to 100% for all shields in Grim Dawn.

Block Recovery Time
The minimum time in seconds between blocked attacks.

Item Parameters

Item Name Tag
A tag reference for the item's name.

Item Quality Tag
A display-only tag reference that goes in front of the item name.

Item Style Tag
A display-only tag reference that goes in front of the item name.

Attribute Scale Percent
A % multiplier for all offensive damaging stats on the item, whether on the base item or on its Affixes. Useful for rolling higher Affix bonuses on one type of item versus another (example: melee weapons vs. ranged weapons) without the need to create additional copies of the same Affixes.

Item Set Name
Record reference to a DBR that contains data for a Loot Set, what set bonuses it has and what items are in the set. If an item is in a set, it must also reference the set here or else the set bonuses will not appear in its tooltip.

Item Cost Name
Record reference to a DBR where various item values are calculated based on the item's type.

Item Text
Additional text displayed underneath the item name, used for flavor text.

Item Classification
The Item's quality (Common, Magic, Rare, Epic, Legendary, Quest). For display purposes only as it does not affect the item's inherent stats or drop rate.

Item Cost
This field is left blank if the Item Cost Name DBR is assigned as that overrides this. If you wish to set a custom value for this item only, it is set here.

Item Level
The item's level, which is important for loot tables.

Cannot Pick Up Multiple
A true/false check for whether the player can have multiples of this item in their inventory. Mainly used for quest items.

Cannot Pick Up
A true/false check for whether the item can be picked up. For when you really hate people who play your mod.

Soulbound
A true/false check for whether the item can be dropped, traded or placed in the Transfer Stash.

Untradeable
A true/false check for whether the item can be dropped or traded. Unlike Soulbound, Untradeable items can still be placed in the Transfer Stash.

Item Requirements
The only field relevant here is the Level Requirement as the rest are controlled by the DBR assigned in the Item Cost Name. If you wish to set a custom requirement for this item only, then you can do so here.

Quest Marker
This is where you set what quests, if any, the item is marked for on the map. You need to know the quest File and the task ID. If the item is relevant for more than one task in a quest, it needs multiple entries.

The Marker Range determines how many meters away the player can be before the item appears on the map.

Racial Bonus
This section is used to give the item bonuses against specific game races. You can apply this bonus to more than one race, but the value has to be the same for all of them. The default game races are as follows:

- Race001 = Undead
- Race002 = Beastkin
- Race003 = Aetherial
- Race004 = Chthonic
- Race005 = Aether Corruption
- Race006 = Bloodsworn
- Race007 = Eldritch
- Race008 = Insectoid
- Race009 = Human
- Race010 = Construct

- Race011 = Riftspawn
- Race012 = Beast
- Race013 = Magical
- Race014 = Celestial
- Race015 = Arachnid

Skill Augment

This section is used to grant bonuses to Mastery skills and to bestow item skills.

Augment Skill Name #

This is a record reference to a mastery skill. If the skill is not in an existing player skill mastery, it will show up as invalid in the item's tooltip.

Augment Skill Level #

The skill rank bonus to the augmented skill.

Augment Mastery Name #

This is a record reference to a player mastery. It must reference the mastery's Class Training DBR.

Augment Mastery Level #

The skill rank bonus to all of the skills in the assigned mastery.

Augment All Level

The skill rank bonus to all of the player's mastery skills.

Item Skill Name

If the item bestows a skill, the record is assigned her.

Item Skill Level Equation

The formula determining the rank of the assigned item skill. This should generally be a formula for Affixes (example: itemLevel/4+1), but can be a fixed value for Unique items.

Item Skill Auto Controller

If the item is cast automatically rather than a granted skill the player can place on their quickbar, the controller record is set here. The default item skill controllers are in Records/Controllers/ItemSkills/.

Pet Bonus

This is a record reference to a pet bonus DBR. Pet bonus DBRs use a custom template but have the same types of attributes as player items.

Offensive Parameters
This section is where all attributes that deal with damage are collected. Of particular importance is Item Base Damage for Caster Weapons (you can set base magic damage here that is not affected by the hardcoded stat variation, known as Jitter). The other important stat is the Offensive Physical sub-section. If flat Physical Damage is set on Weapons or Shields, it will become that item's base damage value, which is not affected by Jitter.

For weapons, bonus flat Physical damage is set under the Offensive Bonus sub-section.

Retaliation Parameters
This section is where all attributes that deal with damage retaliation are collected.

Defensive Parameters
This section is where all attributes that deal with player defenses and damage reflection are collected. Of particular importance is the Defensive Protection sub-section. If a flat Protection value is set on Armor, it will become that item's base armor value, which is not affected by Jitter.

For armor, bonus flat Protection is set under the Defensive Bonus Protection sub-section.

Character Parameters
This sub-section is where all player stats, regeneration, offensive/defensive ability, item requirement reduction and speed bonuses are collected.

It is important to never set the characterRunSpeed/AttackSpeed/SpellCastSpeed values on items as those values are actually decimals and used exclusively for adjusting base speed values on players and monsters. Instead, use the Modifier fields.

Conversion Parameters
This section is where you set an item's Damage Conversion bonuses. An item can only convert one damage type to one other damage type. If the Conversion Out field is left blank, it will act as Multiplicative Damage Reduction.

Skill Parameters
This section is where you can set Cooldown and Energy cost reduction as well as modify the speed of ranged weapon attack projectiles.

Loot Chests
Loot Chests have two important parameters:

Loot Table
This is a record reference to a chest loot table. It is an array so you can override the loot table by difficulty (1 = Normal, 2 = Elite, 3 = Ultimate).

Loot Classification

This classification determines which multipliers are used to determine the quality of its loot.

Item Loot Tables

Loot Tables are organized in a hierarchy. A loot table higher in the hierarchy should never be referenced in a loot table beneath it.

Chest Loot Table

Name	Class	Type	breakableloot_all_a01.dbr
ActorName	Variable	String	
Class	Static	String	
FileDescription	Variable	String	
numSpawnMinEquation	Variable	Equation	numberOfPlayers*1
numSpawnMaxEquation	Variable	Equation	numberOfPlayers*1
loot1Chance	Array	Real	0.1;0;0;0.1;0;0;0;0
loot1Name1	Array	DBR	records/items/loottables/mastertables/mt_accessories_c01.dbr
loot1Weight1	Variable	Int	2
loot1Name2	Array	DBR	records/items/loottables/mastertables/mt_geararmor_c01.dbr
loot1Weight2	Variable	Int	68
loot1Name3	Array	DBR	records/items/loottables/mastertables/mt_gearweapons_c01.dbr
loot1Weight3	Variable	Int	30
loot1Name4	Array	DBR	records/items/loottables/mastertables/mt_accessories_d01.dbr
loot1Weight4	Variable	Int	1
loot1Name5	Array	DBR	records/items/loottables/mastertables/mt_geararmor_d01.dbr
loot1Weight5	Variable	Int	3
loot1Name6	Array	DBR	records/items/loottables/mastertables/mt_gearweapons_d01.dbr
loot1Weight6	Variable	Int	2

Chest Loot Tables are only referenced by chests. They can have Master Tables, Dynamic Tables of Items assigned in their loot pool.

Number Spawn Min/Max Equation

The formula for number of items the chest will dispense. Typically this is a formula based on the number of players

Loot # Chance

An array of weights for rolling this Loot Set. Each entry in the array is an item. If the chest can dispense up to 2 items + 2 * the number of players, then the array should go up to at least 10.

Loot Name 1-6

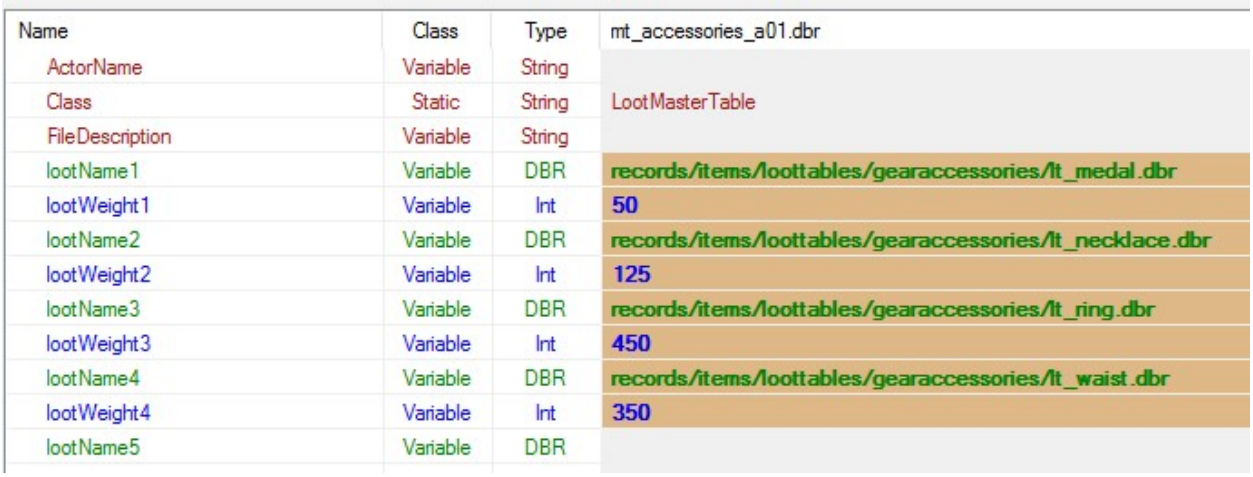
The possible loot from this Loot Set.

Loot Weight 1-6

The relative weights of rolling loot within the Loot Set.

Master Table

Master tables combine Level tables and are the highest level of loot randomization in Grim Dawn that can be assigned to monsters, quests and chest loot tables.



Name	Class	Type	mt_accessories_a01.dbr
ActorName	Variable	String	
Class	Static	String	LootMasterTable
FileDescription	Variable	String	
lootName1	Variable	DBR	records/items/loottables/gearaccessories/lt_medal.dbr
lootWeight1	Variable	Int	50
lootName2	Variable	DBR	records/items/loottables/gearaccessories/lt_necklace.dbr
lootWeight2	Variable	Int	125
lootName3	Variable	DBR	records/items/loottables/gearaccessories/lt_ring.dbr
lootWeight3	Variable	Int	450
lootName4	Variable	DBR	records/items/loottables/gearaccessories/lt_waist.dbr
lootWeight4	Variable	Int	350
lootName5	Variable	DBR	

Level Table

Level tables control at what level a given Dynamic Table starts to be distributed. It has two arrays, one for the levels and another for the Dynamic Tables. Each entry in the array corresponds to the same entry in the other array.

Name	Class	Type	lt_hands.dbr
ActorName	Variable	String	
Class	Static	String	LevelTable
FileDescription	Variable	String	
levels	Array	Int	1;22;40;55
records	Array	DBR	records/items/loottables/gearhands/tdyn_hands_a01.dbr;records/items/loottable...

Dynamic Table

Dynamic Tables are the lowest degree of loot randomization, but also the most important. This is where individual items are randomized and assigned Affixes.

- [Show All]
- All Groups
  - Header
  - Config
  - Loot
  - Randomizer Prefix
  - Randomizer Suffix
  - Randomizer Broken
  - Randomizer Rare Prefix
  - Randomizer Rare Suffix

Name	Class	Type	tdyn_hands_a01.dbr

Config

Name	Class	Type	tdyn_hands_a01.dbr
bellSlope	Array	Real	100;100;80;80;50;50;30;30
minItemLevelEquation	Variable	Equation	(parentLevel*.95)-8
maxItemLevelEquation	Variable	Equation	(parentLevel*1)+5
targetLevelEquation	Variable	Equation	(parentLevel*1)-2
disableLevelLimits	Variable	Bool	
bothPrefixSuffix	Variable	Int	4000
rareBothPrefixSuffix	Variable	Int	4
rarePrefixNormalSuffix	Variable	Int	108
normalPrefixRareSuffix	Variable	Int	92
noPrefixNoSuffix	Variable	Int	25000
prefixOnly	Variable	Int	6500
rarePrefixOnly	Variable	Int	176
suffixOnly	Variable	Int	6500
rareSuffixOnly	Variable	Int	150
brokenOnly	Variable	Int	60000

The Config has several important parts:

- Bell Slope: An array of multipliers for items based on their level. If an item's level is equal to the target level, then it uses the first multiplier, then the next closest item level uses the second entry, etc.
- Min/Max Item Level Equation: The equations for the minimum and maximum item level that can be generated from this loot table. If no items in the loot table fit within these parameters, the loot table generates no loot.
- Target Level Equation: The equation for the target level the loot table is trying to generate. This is coupled with the Bell Slope.
- Disable Level Limits: A true/false check for whether items generated from this loot table need to fall within the parameters of the min/max item level equations and the minimum level enforced by a Level Table. If set to True, items of any level can be generated by this loot table. This is mainly important for potion tables and other tables that may have items of a level below that of a Level Table that dispenses them.
- Affix Weights: These are relative weights of generating combinations of Affixes on the items from this loot table. Broken items automatically disintegrate in Grim Dawn.

Loot

Name	Class	Type	tdyn_hands_a01.dbr
lootName1	Variable	DBR	records/items/gearhands/a01_hands01.dbr
lootWeight1	Variable	Int	1040
lootName2	Variable	DBR	records/items/gearhands/a01_hands02.dbr
lootWeight2	Variable	Int	560
lootName3	Variable	DBR	records/items/gearhands/a02_hands01.dbr
lootWeight3	Variable	Int	780
lootName4	Variable	DBR	records/items/gearhands/a02_hands02.dbr
lootWeight4	Variable	Int	420
lootName5	Variable	DBR	records/items/gearhands/a03_hands01.dbr
lootWeight5	Variable	Int	650
lootName6	Variable	DBR	records/items/gearhands/a03_hands02.dbr
lootWeight6	Variable	Int	350

The Loot section is where individual items and their relative weights are set.

Randomizers

Name	Class	Type	tdyn_hands_a01.dbr
prefixTableName1	Variable	DBR	records/items/lootaffixes/prefix/prefixtables/prefixa01_base_armor.dbr
prefixTableLevelMin1	Variable	Int	1
prefixTableLevelMax1	Variable	Int	500
prefixTableWeight1	Variable	Int	1300
prefixTableName2	Variable	DBR	records/items/lootaffixes/prefix/prefixtables/prefixa01_resistslow.dbr
prefixTableLevelMin2	Variable	Int	1
prefixTableLevelMax2	Variable	Int	500
prefixTableWeight2	Variable	Int	700
prefixTableName3	Variable	DBR	records/items/lootaffixes/prefix/prefixtables/prefixa01_resistabsorption.dbr
prefixTableLevelMin3	Variable	Int	1
prefixTableLevelMax3	Variable	Int	20
prefixTableWeight3	Variable	Int	50

The Randomizers are where Affix tables are assigned. The Randomizers do not distinguish between Prefixes and Suffixes so be sure to only assign tables with Prefixes to the Prefix Randomizers and Suffixes to the Suffix Randomizers, otherwise your items could end up with two Prefixes or two Suffixes.

The Table Level Min/Max controls at what levels the given Affix table can be rolled. The Table Weight is the relative weight of selecting a given Affix table over the others.

Finishing Up

Compile your mod and you can place the item in your level directly, or have it drop via monsters or loot chests if assigned to their loot pool. You are all set!

Tutorial 07: Lua Scripting Basics

This tutorial covers the basics of Lua scripting in Grim Dawn and is not intended as a proper tutorial for Lua or programming. There are many great resources for learning to code/script that are available online and we recommend checking them out before tackling Lua in Grim Dawn.

This tutorial outlines various C++ hooks created for Lua scripting in Grim Dawn.

Adding New Scripts for Mods

For tools purposes, it is important that your mod has its own version of main.lua in “source/Scripts/” as an entry point. Typically, main.lua will load “scripts/libs/shared.lua” and “scripts/game/grimdawn.lua”, but you can overwrite what grimdawn.lua loads within your mod.

Granting Multiplayer Quests

For NPC Conversations, only the talking player can accept quests or trigger scripted events. This is important for quest decisions as multiple players speaking to the NPC at the same time could otherwise trigger conflicting events (example: an NPC going hostile or remaining friendly). However, you may still want to bestow quests and quest decisions in Multiplayer to the other players.

For this, you will need to set up scripts that bestow the quest after checking for the same conditions that the speaking player had to go through within the dialogue. This is where knowing all of your quest and task IDs is very important.

```lua
-- A Life for an Eye
local lifeForAnEyeQuest = false
local lifeForAnEyeReturnQuest = false

function gd.MPQuestControl.bestowLifeForAnEyeQuestGlobalMP()

    if not lifeForAnEyeQuest then
        lifeForAnEyeQuest = true
        local player = Game.GetLocalPlayer()

        local questId = 0x68425D80
        local taskId = 0x1C5AA340
        local questId02 = 0x10C36380
        local taskId02 = 0xA0B58800

        local questTaskState = player:GetQuestTaskState(questId, taskId)
        local questTaskState02 = player:GetQuestTaskState(questId02, taskId02)

        -- Begins first task of Life for an Eye, checks if player has completed Disarming the Enemy and is not already on the quest
        if (questTaskState == QuestState.Complete && questTaskState02 != QuestState.InProgress && questTaskState02 != QuestState.Complete) then
            player:GrantQuest(0x10C36380, 0x8BBAFB00)
        end

    end

end
```

In this example, the player must have completed a prerequisite quest, but because the quest can be completed retroactively, you also want to check for whether the second task was finished.

If your quest can be completed retroactively, you will also need to test for a new set of conditions and bestow the Turn In step instead:

```
function gd.MPQuestControl.bestowLifeForAnEyeReturnQuestGlobalMP()

  if not lifeForAnEyeReturnQuest then
    lifeForAnEyeReturnQuest = true
    local player = Game.GetLocalPlayer()

    local questId = 0x68425D80
    local taskId = 0x1C5AA340
    local questId02 = 0x10C36380
    local taskId02 = 0x8BBAFB00

    local questTaskState = player:GetQuestTaskState(questId, taskId)
    local questTaskState02 = player:GetQuestTaskState(questId02, taskId02)

    -- Begins second task of Life for an Eye, checks if player has completed Disarming the Enemy and has MONEYBAGS_SLAIN token
    if (questTaskState == QuestState.Complete && questTaskState02 != QuestState.InProgress && player:HasToken("MONEYBAGS_SLAIN")) then
      player:GrantQuest(0x10C36380, 0xA0B58800)
    end

  end

end
```

In this example, the player must have completed a prerequisite quest, is not already on the first task of the quest (an unlikely scenario, but could happen in an odd combination of Multiplayer sessions) and has a token confirming that they killed the boss before.

QuestEvent(string)

This function is used to execute a specific function from the Quest Event list that is then run by the Host. This is important for executing a scripted event that affects the game world but may not necessarily have been triggered by the Host player.

Example: a Client enters a trigger area that spawns an enemy based on the Host's earlier quest choices. As the world state is based on the Host, QuestEvent is run so that the spawn is executed by the Host (which in turn spawns the monster for all players), rather than by the Client.

QuestGlobalEvent(string)

This function is used to execute a specific function from the Quest Event list that is then run by all players in a Multiplayer session. This is important for granting quests in Multiplayer or for bestowing Tokens to all players based on one player's actions.

Quest Event List

Scripts intended to be run in Multiplayer, either by the host or by all players, are cataloged in questevents.Lua.

Example: repairWightmireBridgeToken = gd.quests.burrwitchBridges.wightmireBridgeRepairGlobalMP,

If QuestGlobalEvent("repairWightmireBridgeToken") is then run, all players in the Multiplayer session will run the function, which bestows a Token for that tracks whether the bridge was restored by the player.

Grim Dawn Script Reference

Static methods can be called directly on the class (example: Game.GetLocalPlayer()). Instance methods require an instance of the class to operate on. To get an instance, often, Class.Get(id) is used (example: Game.GetLocalPlayer():GetPlayerName() returns the name of the local player). Constructors can be used to create a new instance of a class.

The format for static methods:

```
ReturnType Class.MethodName(ArgType1 argName1, ArgType2 argName2, ...)
```

The format for instance methods:

```
ReturnType instance:MethodName(ArgType1 argName1, ArgType2 argName2, ...)
```

Methods with no return type do not include one.

The format for instance variables:

```
VariableType instance:VariableName
```

Comments and Operators

```
-- Single-line comment

/* Multi-line

comments
```

```
*/

!=  Not-Equal operator, rather than Lua's standard ~=

&&  AND operator, rather than Lua's standard "and"

||  OR operator, rather than Lua's standard "or"

'If Server then'  this check is performed at the beginning of functions that impact the world state as a precautionary measure so that only the Host is executing them.
```

Token Checks

```
GiveTokenIfPlayer(id, token)

GiveTokenToLocalPlayer(token)

RemoveTokenFromLocalPlayer(token)
```

State Based Object Swap

This series of functions has been written to handle the swapping of objects and entities in the world based on the Host player's Tokens. Script Entities run these functions upon loading and unloading. The Update function is called independently to update the state of a Script Entity that is already loaded (example: an NPC the player is talking to switches to a hostile state).

```
TokenStateBasedObjectSwap(objectid, userdata, states, cls, argsfn, MPOverrideState, MPOverrideDbr)

UpdateObjectSwap(objectid, states, cls, argsfn, MPOverrideState, MPOverrideDbr)

ClearObjectUserdata(objectid)
```

- 'id' is the id of the script entity
- 'userdata' is a table containing the current state and object
- 'states' is an ordered table containing a mapping of tokens (in order of precedence) to state/dbr pairs
- 'class' is the class used to create the entity
- 'argsfn' is a function used to pass extra arguments to the class create method
- 'MPOverrideState' boolean determining whether the MPOVerrideDbr should be used. If true, the override is used instead of the host's current token state.
- 'MPOverrideDbr' is a Dbr override used in the case that a client is on a quest the host is not that requires an entity state change (such as an NPC going hostile)
- Returns 2 values:
  'didWork' (boolean) - True if a swap occurred based on changes in tokens or MPOverrideState.
  'newState' - The new state of the object is a swap occurred, or the current state if one didn't.

Example of the Load/Unload functions for a bridge:

```
local oldGroveBridgeStateObjects = orderedTable()
    oldGroveBridgeStateObjects["GD_OLDGROVEBRIDGE_REPAIRED"]      = { state = BridgeState.Repaired,  dbr = "records/ston
    oldGroveBridgeStateObjects[""]                                = { state = BridgeState.Destroyed, dbr = "records/storyelements/questa

function gd.quests.devilsCrossingBridges.oldGroveBridgeOnAddToWorld(objectId)

    if Server then

        oldgrove.bridgeId = objectId

        local userdata = {}
        TokenStateBasedObjectSwap(objectId, userdata, oldGroveBridgeStateObjects)
        Shared.setUserdata(objectId, userdata)

    end

end

gd.quests.devilsCrossingBridges.oldGroveBridgeOnDestroy = ClearObjectUserdata
```

Example of the bridge being updated dynamically after being repaired:

```
function gd.quests.devilsCrossingBridges.farmlandBridgeRepair()

    if Server then
        local blocker = Entity.Get(farmland.blockerId)

        QuestGlobalEvent("repairFarmlandBridgeToken")

        if (blocker != nil) then

            local coords = blocker:GetCoords()
            local pfx = Entity.Create("records/fx/general/bridgerepair_fxpak01.dbr")

            if (pfx != nil) then
                pfx:NetworkEnable()
                pfx:SetCoords(coords)

            end

        end

        UpdateObjectSwap(farmland.bridgeId,  farmlandBridgeStateObjects)
        UpdateObjectSwap(farmland.blockerId, farmlandBridgeBlockerStateObjects)

    end

end
```

Game

```
// static methods

Player Game.GetLocalPlayer()

int Game.GetNumPlayers()

Player Game.GetPlayer(int index)

int Game.GetMinPlayerLevel()

int Game.GetMaxPlayerLevel()

int Game.GetAveragePlayerLevel()

unsigned int Game.GetGameTime() // returns the time in ms that the game has been running
```

Script

```
// static methods

Script.Load(string path)
```

Debug

```
// static methods

Debug.AddStatisticText(string text)

Debug.Log(string message)
```

Shared

```
// static methods

string Shared.Localize(string tag)

string Shared.Localize(string tax, arg1, arg2, ..., argN) // unimplemented
```

Player

```
// static methods

Player Player.Get(objectId)

bool Player.IsType(objectId)
```

// instance methods

string player:GetPlayerName()

bool player:HasToken(string token)

player:GiveToken(string token)

player:RemoveToken(string token)

player:DestroyItem(string dbrName)

--- Quests

// Note that quest and task ids should be prefaced with 0x to denote hex (example: 0xBFE25500)

QuestState player:GetQuestState(uint32 questUid)

Returns:

QuestState.Invalid, no such quest

QuestState.Unavailable, no tasks are currently available to the player

QuestState.Available, one or more quest tasks are available to the player

QuestState.InProgress, one or more quest tasks are currently in progress

QuestState.Complete, all quest tasks have been completed by the player

QuestState player:GetQuestTaskState(uint32 questUid, uint32 taskUid)

Returns:

QuestState.Invalid, no such quest or task

QuestState.Unavailable, task not currently available (doesn't meet conditions, etc)

QuestState.Available, task is currently available

QuestState.InProgress, player is on the task

QuestState.Complete, player has completed the task

```
void player:GrantQuest(uint32 questUid, uint32 taskUid)


--- Factions

FactionValue player:GetFaction(string factionEnum)

Returns:

Int factionStatus, numerical value of the player's faction state
```

Actor

```
Actor Actor.Get(objectId)
```

Door

```
void Door.Open()

void Door.Close()

void Door.SetLocked(bool locked)
```

Lever

```
void Lever.SetLocked(bool locked)
```

Shrine

```
void Shrine.SetLocked(bool locked)
```

Destructible

```
void destructible:Destroy(Vector direction, float force, bool dropLoot)
```

Character

```
// enums

SoundType.VoxSound

SoundType.SpecialAttackSound1
```

```
SoundType.SpecialAttackSound2

SoundType.SpecialAttackSound3

SoundType.SpecialAttackSound4

SoundType.GenericSound1

SoundType.GenericSound2

SoundType.GenericSound3

SoundType.GenericSound4

SoundType.VoiceSound1

SoundType.VoiceSound2

SoundType.VoiceSound3


// static methods

Character Character.Get(objectId)

bool Character.IsType(objectId)


// instance methods

character:AdjustMoney(int amount)

character:GiveExperience(unsigned int amount)

character:GiveLevels(unsigned int amount)

character:GiveSkillPoints(unsigned int amount)

character:Attack(targetId)

character:PlaySound(SoundType sound)

character:Move(WorldVec3 position)

character:Turn(Vec3 direction)

character:Kill()
```

character:GiveItem(string itemDbr, int amount, boolean complete), complete only used for components, false for all other items.

character:TakeItem(string itemDbr, int amount, boolean complete)

character:HasItem(string itemDbr, int amount, boolean complete), returns boolean

Entity

```
// static methods

Entity Entity.Get(objectId)

Entity Entity.Create(string dbrPath)

bool Entity.IsType(objectId)


// instance methods

WorldCoords entity:GetCoords()

entity:SetCoords(WorldCoords coords)
```

Object

```
// static methods

Object Object.Get(objectId)


// instance methods

int object:GetId()

string object:GetName()

object:Destroy()
```

Proxy

```
// static methods

Proxy Proxy.Get(objectId)
```

```
Proxy Proxy.Create(string dbrPath)

bool Proxy.IsType(objectId)


// instance methods

bool proxy:AllKilled()

proxy:Run()
```

WorldCoords

```
// properties

[ C ] Vec worldCoords:origin

[ C ] Vec worldCoords:xAxis

[ C ] Vec worldCoords:yAxis

[ C ] Vec worldCoords:zAxis
```

Vector

```
// constructor

[ C ] Vector(float x, float y, float z)


// properties

[ C ] float vector:x

[ C ] float vector:y

[ C ] float vector:z


// methods

[Lua] float vector:Length()  // Returns the length of the vector

[Lua] float vector:LengthXZ()  // Returns the length of the vector on the XZ plane
```

```
[Lua] float vector:LengthSquared()  // Returns the squared length of the vector

[Lua] float vector:LengthSquaredXZ()  // Returns the squared length of the vector on the XZ plane

[Lua] Vec   vector:Unit()  // Returns a unit length copy of the vector

[Lua] float DotProduct(Vec a, Vec b)  // Returns the dot product of a pair of vectors

[Lua] Vec   CrossProduct(Vec a, Vec b)  // Returns the cross product of a pair of vectors
```

Time

```
// methods

[ C ] uint32 Time.Now()  // Returns time since system start in milliseconds
```

UI

```
// methods

UI.Notify(string Tag)  // Sends a notification to players's screen using the defined Tag
```

Randomization

```
// methods

math.randomseed(Time.Now())  // Generates a new seed for the function

Int random(int min, int max)
```

Userdata

```
// methods

Shared.setUserdata(objectId, userdata) // userdata is a table containing variables assigned to the given objectId. This data is stored outside of the function for reference and is not wiped when the function finishes running.

Shared.getUserdata(objectid) // used to load the userdata stored for the given objectId
```

Repeating Event

```
// methods

Script.RegisterForUpdate(objectId, String functionName, updatePeriod)  // objectId has to be a script entity that handles the event, the functionName is a string reference, the updatePeriod is how often the function runs, in milliseconds.

Script.UnregisterForUpdate(objectId, string functionName) // halts the repeated running of the function

Shared.restoreUserdata(objectId, string eventKey) // eventKey is a string reference for loading the same event's persistent data.

Shared.persistUserdata(objectId, string eventKey)
```