Upgrading to Barotrauma v0.9 - Guide to modders

Version 0.9 is by far the biggest update to Barotrauma - over a year in the making, it includes changes to almost all of the core mechanics of the game, adds a huge number of new features and significantly expands the modding possibilities. We've strived to maintain backwards compatibility with mods and submarines created for previous versions of Barotrauma as much as we can, but there are still some things that should be changed or tweaked to make them work smoothly with the new version.

This guide is intended to help Barotrauma modders upgrade their mods to the new version, and it's written with the assumption that the readers are familiar with how modding and the game's systems worked in the previous versions.

There are most likely things we've forgotten to include here, and things we should explain in more detail. If you have any questions you can't find an answer to here, run into any issues or have any suggestions regarding the new features, please let us know! You can contact us through any of the following:

Discord

https://discordapp.com/invite/xjeSzWU

GitHub

https://github.com/Regalis11/Barotrauma/issues

Email

contact@barotraumagame.com

Identifiers

Older versions of Barotrauma used names to identify things such as items, jobs and skill levels. This turned out to be a bad idea, because it made it very difficult to translate the game, change the names of items or replace items with something else without breaking all existing submarine files and many other things.

From version 0.9 onwards almost everything uses identifiers - unique strings that aren't shown to the players but are used to identify things under the hood. In the vanilla files the identifiers tend to be simply lower-case versions of the name without spaces - the identifier of a Welding Tool is "weldingtool" and so on.

When porting your mod to v0.9, you have to add identifiers to each item your mod adds. So, instead of

```
<Item name="Medical Syringe">
```

You would have

```
<Item name="Medical Syringe" identifier="medicalsyringe">
```

You also need to go through the item configurations which refer to other items by their name. For example, if you have a status effect that requires the item to be contained inside a Medical Syringe, you would use `identifier="medicalsyringe"` instead of `name="Medical Syringe"`.

You can make sure you've replaced the names with identifiers in every file by simply starting up the game. If something is still using names, the console will pop up and show an error message. If your mod adds new jobs for the crew, you should also make sure all of the jobs have an identifier and that the skills are defined using identifiers.

Content Packages

Content packages function pretty much the same as they did in previous versions. However, there's one important difference: it's now possible to enable multiple content packages at the same time, making it much easier to use multiple mods at the same time.

Content packages can also be configured as "core packages". These are packages that contain files that are required for the game to run, such as the executables and important configuration files the game doesn't work without. There must always be one (and only one) active core package. You should only configure your mod as a core package if it's intended to replace or remove a large portion of the vanilla files. Mods that only add new content to the vanilla game (or to other mods) should not be core packages.

Another thing to note is that content packages are not allowed to add files outside the "Mods" folder when installing them from the Steam Workshop. So if your mod has previously added files directly to the content folder, you should instead make your own subfolder inside "Mods" and make the file paths in the content package point there. The only exception are submarine files, which can still be placed in the "Submarines" folder.

Inventory icons

Items can now have a separate sprite that's shown in the inventory when the item has been picked up. As the in-game sprites are usually quite small and low-resolution, you may want to add icons to your new items using:

```
<InventoryIcon texture="SomeIconFile.png" sourcerect="x,y,width,height"/>
```

Sprites

We now use higher-resolution sprites and allow the camera to zoom in a little bit closer. If you don't want your item/character sprites to appear blurry and pixelated, you should double their resolution and scale the item/character down. Items can be scaled by adding scale="0.5" to the item element in an XML file, and characters can be scaled in the character editor.

We've also added the option to make items use multiple sprites, with some simple animation options and a way to modify the sprites/animations based on the item's state or properties. This can be used for simple things such as cabinets with a glass door that's drawn in front of the items inside the cabinet, or more complex stuff like devices with parts that start moving when the device is powered. The additional sprites can be defined using a DecorativeSprite element - you can take a look at items such as the Vent, Alien Motion Sensor and the Oxygen Generator to see how they're configured.

It's also now possible to change item sprites according to the container they're inside (for example, a uniform could look like a bundle of cloth when laying on the floor, and hang neatly when placed in a coat hanger). There's a configured using a "ContainedSprite" element, which is otherwise identical to the sprite elements, but also has a "allowedcontaineridentifiers" or "allowedcontainertags" attribute. The attributes determine which identifier/tag the container must have for the sprite to become active.

Lighting system

There's been a major change to the lighting algorithm in version 0.9 that should be taken into account if your mod includes items with lights. In earlier versions a pure white light would only make the sprites behind it appear at their original brightness, but never make them any brighter. As of version 0.9, full-brightness lights make things brighter and may cause the areas around them to appear overexposed (which may be a desired effect if you're making something like a flashbang or a high-intensity floodlight!).

In general, the brightness of all lights should be dimmed down to make them look the same as they did in previous versions. You can get them to look roughly the same by dividing the color values or the alpha by 2 (for example, 255,255,255,255 would become either 127,127,127,255 or 255,255,255,127). There's a console command called "multiplylights" that can be used to adjust all the lights in the submarine at once. Executing "multiplylights 1,1,1,0.5" would make all the lights half as bright.

Afflictions

Version 0.9 adds many new features to the health system. One of the key aspects of the new system are afflictions, which can be things such as internal damage, blood loss, drunkenness, mental disorders, hunger, infections…

Afflictions replace the damage types in the old versions (damage, burn, stun) and all status effects that cause (or heal) some type of damage should be updated.
The following status effect

```
<StatusEffect type="OnUse" target="Character" damage="-10" stun="5"/>
```

Would be updated as follows:

```
<StatusEffect type="OnUse" target="Character">
  <ReduceAffliction identifier="internaldamage" amount="10.0"/>
  <Affliction identifier="stun" amount="5.0"/>
</StatusEffect>
```

You also need to make the item usable in the health interface by adding `useinhealthinterface="true"` to the item element in the XML.

Old medical items and their status effects still work for the most part, but it may be desirable to update them to use the affliction system. For instance, you might have had a poison item with a status effect that causes damage for 60 seconds, and an antidote item that removes said status effect. This will still work, but the affliction system offers way more possibilities in what you can do:

- You can define an icon, name and description for the poisoning affliction. These will be visible in the health interface. You can also NOT display them if you want to add a more hard-to-detect affliction.
- You can have custom "cause of death" descriptions for the afflictions.
- There are several adjustable screen distortion effects available for the status effects (barrel distortion, chromatic aberration, blur, a wavy distortion effect). See Afflictions.xml for examples.
- The affliction can have different effects depending on the strength of the affliction. For example, a hunger affliction could initially do nothing, start slowing down the character

when it starts to get high enough, and finally start decreasing health and cause the screen to distort if the character doesn't find something to eat.
- Afflictions can also affect NPC dialogue. You can define a special "dialog flag" for the affliction, and add new NPC lines that can only be said by characters with the specified flag.

Repairables

There's been some changes to the way items are repaired. Instead of the old "FixRequirements" there's now an item component called "Repairable" which defines how the item can be repaired. You can see some examples of how the Repairables are configured by taking a look at the Nuclear Reactor or Junction Box items. There's a few differences to the old FixRequirements:

- Any item can be repaired by any character, but a character with insufficient skill levels will take longer to do the repairs. The durations are configured using the attributes "FixDurationLowSkill" and "FixDurationHighSkill".
- Items can be configured to deteriorate over time. The intention is to give the Engineers and Mechanics more work to do even if the round is running smoothly. The relevant attributes are "DeteriorationSpeed", "MinDeteriorationDelay", "MaxDeteriorationDelay" and "MinDeteriorationCondition".
    - DeteriorationSpeed: Determines how fast the item's condition decreases (in units per second, 0.1 would mean the condition decreases by 6 per minute).
    - MinDeteriorationDelay, MaxDeteriorationDelay: The delays can be used to prevent all of the items from breaking at the same time: each item gets assigned a random delay between the min and max values, and it won't start deteriorating until that delay passes.
    - MinDeteriorationCondition: prevents the item from spontaneously deteriorating below the specified condition. For example, you might not want Junction Boxes to spontaneously go all the way down to zero and stop working. By setting MinDeteriorationCondition to 10, they would deteriorate down to 10 and only go down to zero if something else does additional damage to them.
- You can configure special particle effects for damaged items. For example, a junction box could emit a bit of sparks when it's partially damaged, and start smoking when it's completely broken. You can take a look at the Nuclear Reactor and Junction Box items to see how these are defined.

Characters

There's been quite a lot of changes to the way characters are configured. We now have a in-game character editor that makes it much easier to create and edit characters and their animations.

Unfortunately getting old character files imported into the character editor requires quite a bit of work - for the time being it may be fastest and easiest to simply create a new character with the editor using the spritesheet of the old character, or if the character is similar enough to some existing creature, you could copy it, swap the graphics and tune it in the editor.

As of now we don't have full documentation or tutorials written for the character editor, but we believe modders who've created characters using the xml configuration files will find it somewhat easy to get a hang of: you're essentially just editing the same values you did before, but now you can do it using a visual tool that immediately shows what kind of an effect the changes have to the character or animations.

We will provide better documentation for the editor in the future, but for the time being these notes may help you get started:

Character configurations are now split into several xml files:

1. The "base file" which includes things like the AI, health, inventory and particle settings of the character. This is pretty similar to the old character configuration files, except that the ragdoll and animation parameters are not included in it anymore.
2. Ragdoll files. At the moment we only support one ragdoll file per character type, but we're planning on making it possible to define multiple ones, so there can be a bit of variation in what the characters/monsters look like. The ragdoll configuration file(s) should be placed inside a folder called "Ragdolls", which in turn should be located in the folder where you have the base configuration file. The name of the ragdoll file should be "[Character]DefaultRagdoll.xml" (replace [Character] with the name of the character).
3. Animation files. At the moment we support four types of animation configurations: Walk, Run, SwimSlow and SwimFast. These should be placed inside a folder called "Animations", which should be located in the folder where you have the base configuration file. The animation files are named in a similar fashion as the ragdoll file, "[Character]Walk.xml", "[Character]Run.xml", "[Character]SwimSlow.xml" and "[Character]SwimFast.xml".

Random events

The way random events (monster and item spawns) are configured has also changed a bit. Instead of configuring each random event separately and having the game randomly pick a couple of them, we now use what we call "event sets". The game chooses one event set per level based on the difficulty and type of the level. The vanilla event sets are configured in Content/randomevents.xml.

Here's an example of one event set (excluding the actual events included in the set):

```
 <EventSet minleveldifficulty="0" maxleveldifficulty="10">
   <Commonness commonness="5">
     <Override leveltype="RidgeBasic" commonness="0" />
     <Override leveltype="WastesBasic" commonness="0" />
   </Commonness>
```

This event set can appear in levels with a difficulty between 0 and 10 (difficulty being a value between 0 and 100). The commonness of the set is 5 in all levels except RidgeBasic and WastesBasic. The commonness is an unitless value that determines the probability for the event set to be selected relative to other sets. For example, if you had two sets with a suitable difficulty for the selected level, one with a commonness of 1 and another with a commonness of 10, the latter would be 10x as likely to get chosen.

The event sets can have “child event sets”, which can be used to create logic like “spawn 5 crawlers OR spawn a moloch” or “spawn 4 crawlers AND 2 threshers”. Here’s an example of such child event set:

```
   <EventSet chooserandom="true" mindistancetraveled="0.20" minmissiontime="150" minintensity="0.0"
maxintensity="0.25">
     <MonsterEvent characterfile="Content/Characters/Crawler/Crawler.xml" minamount="1"
maxamount="1" spawntype="mainpath" musictype="monster" />
     <MonsterEvent characterfile="Content/Characters/Moloch/Moloch.xml" minamount="1"
maxamount="1" spawntype="mainpath" musictype="monster" />
   </EventSet>
```

This would cause either one crawler or one moloch to be spawned when submarine has passed 20% of the level or after 150 seconds have passed since the start of the level. The chooserandom-attribute causes one of these events to be chosen. If set to false, all of the events would be chosen, causing both of the monsters to spawn.

The minintensity/maxintensity values can be used to delay additional random events when the situation is already bad. The intensity of the current situation is a value between 0 and 1, and it’s calculated based on several factors, including flooding, fires, the amount of nearby monsters or monsters inside the sub, whether the submarine is powered, and the average health of the crew. In this example, the crawler/moloch won’t spawn if the intensity is above 0.25.

You can see how the intensity is calculated in more detail and what kind situations cause the intensity go up by entering “debugdraw” in the debug console while a round is running, and then enabling the event debug information by pressing Ctrl+E.

Misc notes

- It’s now possible to define color values as bytes in the range of 0-255 instead of floats in the range of 0.0-1.0 (e.g. color=”255,127,0,255” instead of “1.0,0.5,0.0,1.0”)

- Monsters can now have inventories, and you can set them to automatically spawn with certain items. This can be used to give the monsters loot that can be accessed when the monster dies - see crawler.xml for an example.

- Characters can have status effects (similar to items). You could, for example, make a monster explode when it dies, have a monster induce psychosis psychosis in nearby characters or give humans a permanent status effect that gradually increases a Hunger affliction.

We’ve probably forgotten to include many things here that we should’ve mentioned, so as said, if you have any questions or run into any problems, please drop as a message!