Banished Toolkit 1.0.7

1. Banished Toolkit Prerequisites

- The Banished toolkit requires Banished version 1.0.4 Beta or better. While the shipped game runs under Windows XP, the toolkit does not. The toolkit was developed and used under Windows 7. Other operating systems have not been heavily tested.
- The toolkit requires all graphics SDKs that the game supports to properly compile shaders. At the moment DirectX 11 and DirectX 9 are required. OpenGL support will be required in new releases.
- The data files from your purchased game are required.
- The DirectX End User Runtime will have to be installed. You can find it here: http://www.microsoft.com/en-us/download/details.aspx?id=35
- The Visual Studio 2012 Redist is also required. It can be downloaded here: http://www.microsoft.com/en-us/download/details.aspx?id=30679 The toolkit may be compiled with newer versions of MSVC in the future - if so you'll have to install the redist for the newer versions.
- For machines (mostly laptops) that have both an Intel card and Nvidia card, make sure you set the tool and game executables to use the high powered video card, rather than the integrated one.
- The game uses the font FrancophilSans. You may need it installed depending on what you are modding.

2. Toolkit Contents

The toolkit has the following structure

```
/bin
   xWMAEncode.exe                        ->    Audio encoding tool (from DXSDK
June2010)
   /WinData                              ->    Location where pack files need to be.
   /x64
        Application-x64-profile.exe      ->    Developer version of the game.
        Tools-x64.exe                    ->    Tool to compile resources, build mods.
        Runtime-x64-profile.dll          ->    Game engine dll
        VideoDX9-x64-profile.dll         ->    DirectX 9 Renderer
        VideoDX11-x64-profile.dll        ->    DirectX 11 Renderer
    CompileWin-x64-profile.dll     ->    Used to compile game resources on windows
    VideoCompiler-x64-profile.dll  ->    Used to compile SRSL shaders
        libfbxsdk.dll                    ->    FBX loader (from FBXSDK)
        cmd.txt                          ->    Command line parameters that are always
used.
   /x32
        Application-x32-profile.exe      ->    Developer version of the game.
        Tools-x32.exe                    ->    Tool to compile resources, build mods.
        Runtime-x32-profile.dll          ->    Game engine dll
        VideoDX9-x32-profile.dll         ->    DirectX 9 Renderer
        VideoDX11-x32-profile.dll        ->    DirectX 11 Renderer
    CompileWin-x32-profile.dll     ->    Used to compile game resources on windows
    VideoCompiler-x32-profile.dll  ->    Used to compile SRSL shaders
        libfbxsdk.dll                    ->    FBX loader (from FBXSDK)
        cmd.txt                          ->    Command line parameters that are always
used.
/example
        /animal                          ->    Example of making a new animated animal
and adding it to a pasture.
        /building                        ->    Example of adding a new building that
takes one resource and makes another.
        /crop                            ->    Example of adding a new farm crop.
        /translation                     ->    Example of font, character sets, and
```

```
translations.
        /tree                           ->    Example of making a tree and adding it
to an orchard.
    /limitresource                  ->    Example of how to use custom resource limits
    /buildrequirement               ->    Example of how to use more than 3 build
requirements for a building.
/resource
        /.                              ->    Lots of game data for examples and
modification
BuildExamples.bat                       ->    Quickly build all the mod kit examples.
BuildResources.bat                      ->    Build all game resources contained in
/resource
```

3. Command Line Parameters

Common command line Parameters

- **/bin <path>** Set the location of the bin folder relative to the executable. This defaults to nothing, but the cmd.txt file sets it to ..\
- **/pathres <path>** Set the resource directory. This directory is relative to /bin. The default is /bin/../resource
- **/pathdat <path>** Set the compiled resource directory. This directory is relative to /bin. The default is /bin/WinData
- **/pathpkg <path>** Set the package directory. This directory is relative to /bin. The default is /bin/WinData

Command line parameters for Application-x64-profile.exe

- **/onlypkg** Only use .pkg (game package) and .pkm (game mod) files when loading. Otherwise anything in the resource folder will be loaded first.
- **/launcher** Start the game with the launcher. You can also hold CTRL as the game starts to bring up the launcher.
- **/nomods** Start the game with all mods disabled. This can be useful if a mod causes a crash and the game can't start.
- **/ref <resource>** Specify an mod resource list to load when starting the game. This can be used to load resources before a mod is packaged. For example: /ref Package.rsc:modName

Command line parameters for Tools-x64.exe

- **/build <resource>** Build a resource and all dependencies. Example /build Game/GameResources.rsc:resource
- **/mod <pkgresource>** Build a mod package. Any changes that differ from original data and new files will be packaged into one file.
- **/texpad <pngfile>** Fill any alpha = 0 pixels in a texture with expanded border color. This is useful for making textures that mipmap nicely.

4. Getting Started

After unpacking the archive, you'll want to test the toolkit environment to make sure everything is working properly.

1. Copy the .pkg files from the game data directory to where ever the toolkit is located. In this example, assume the toolkit it installed at C:\BanishedKit\. You'll copy to C:\BanishedKit\bin\WinData. If you're using Steam, the data files are probably in C:\Program Files (x86)\Steam\SteamApps\common\Banished\WinData. Non Steam builds default to C:\Program Files\Shining Rock Software\Banished\WinData
2. Open a command prompt and change directory to the toolkit. (C:\BanishedKit)
3. Run the batch file BuildResources.bat. This is going to take a few minutes.

4. If all goes well, all resources will be compiled to binary format and be placed in /bin/WinData. You'll see a lot of output along the lines of *Compiled ImageBuffer: Terrain\TerrainDirtTexture.rsc:resource.* Generally the tool will notify of any error with a dialog box or text in the output. Or it might crash.

   If it does have errors or crash, make sure you've got the game .pkg files in the right place and you've installed all the prerequisites.
5. Start the developer build of the game, /bin/x64/Application-x64-profile.exe. The game should run as normal. It's loading the newly compiled resources - which should match what's in the game package file.
6. Check to see if you can modify a resource and have it effect the game.
   1. Open /resource/Dialog/StringTable.rsc in a text editor.
   2. Search for *StringTable mainMenu*.
   3. Look for and change the entry that looks like
      { String _name = "NewGame"; String _text = "New"; }
      to
      *{ String _name = "NewGame"; String _text = "**MODDED!!!**"; }*
   4. Save the file.
   5. Restart Application-x64-profile.exe. The game will notice the change to the string table and recompile it. You could also rerun the BuildResources batch file, but this isn't required - the game is capable of compiling all resources, however you don't get detailed text output as you would on the command line with Tools-x64.exe.
   6. When the game loads, you should see that the new game button now has the modified text.
7. You're now setup to modify the game. Obviously, you'll want to undo your change to the string table.

5. Best Practices

Most of the game configuration data is provided as an example for how everything in the game is put together. You can modify it as you please, however if you plan on distributing your mod to others, there's a few things to keep in mind so that your mod will working well with other mods.

Take care with resources that might be shared!

It's ill advised to make changes to core resources. If multiple mods reference new resources in these files, only one set will be loaded, causing unintended behavior. Some examples of these resources are:

- **Game/Toolbar.rsc** Items that appear on the game toolbar.
- **Dialog/StringTable.rsc** Most all text for the game.
- **Dialog/SpriteSheet.rsc** Sprites that the game uses to render the user interface.

If you need to load new text, sprites, raw materials, natural resources, livestock, etc, you should make your own list of resources and tell the game to load it. You'll make an ExternalList resource. The mod in /example/building has a good example of this.
For example, if you're adding the apiary, bee keeper, and honey, in a file apiaryResources.rsc:

```
ExternalList resource
{
        External _resources
        [
                "Apiary.rsc:apiary"             // this references everything for the
apiary
        ]
}
```

When running the game, start it like so:
Application-x64-profile.exe /ref apiaryResources.rsc

After you package your mod, this file will be loaded automatically when the mod is enabled if it is named correctly.

If you're just modifying text, user interface layout or look, models, textures, settings for citizens, etc you can just override the original resources. If your mod does conflict with another users mod, they will be notified of the conflict if both mods are loaded and enabled at the same time.

Working on multiple mods.

If you have multiple mods you are working on, and as you work through the examples in this document, it's best to either clear out the bin folder of compiled resources when switching between mods, or use the /pathdat command line flag to make sure that binary resources are separated.

When mods are packaged, any file in the current binary folder that differs from the one in the original game packages is put into the package.

Be aware of new materials and audio.

As of version 1.0.5 Beta, materials now use a new shading language. The old format will no longer work and any custom previous shaders built for 1.0.4 will have to be reworked. Old materials from old mods will still load in game and work properly.

If you're going add audio to the game, be aware that at some time in the future, you'll probably have to make changes to support other audio systems as the game is ported to other platforms. If you don't add any new audio, you're mod is probably ok, but may require extra work if you end up working on it when new versions are released.

Sharing Mods

Once you've developed mods, it's best to upload them both to steam workshop and to locations where non-steam users can download your mod. Some of these include http://banishedinfo.com/, and http://www.nexusmods.com/banished/

6. A Note on Save Games

Save games store which mods were enabled when they are saved. When they are reloaded, all mods are first disabled, and then only those that were previously enabled in the save are loaded. When new games start, the mods setup in the main menu are the ones that the game will start with.

While a game is running, you can enable mods, so that you can add new buildings to an already existing map. However if resources that shipped with the game are changed, unexpected behavior might occur. For example if the size on the map of a storage barn changes because of a mod, when old barns are removed, they can cause pathing issues and crashes.

Additionally if you disable a mod while a game is running, crashes can also occur. If you've enabled a mod that adds buildings, and those buildings are placed, disabling the mod will cause a crash.

Save games will also detect changes to a mod. If a new mod version has been published and you've overwritten the mod with the new version, the game will warn you that the mod is different or has changed version and that unexpected behavior can possibly occur.

7. Steam Workshop

Once you've developed a mod, you can upload it to steam workshop to share it with the steam community. To do this, you'll have to use the Steam version of the game. Put your mod in: C:\Program Files (x86)\Steam\SteamApps\common\Banished\WinData\

When you start the game, the mod will show up in the Mod menu. Click on the '...' button for the mod - it's right under the button to enable the mod in game.
Uploading to Steam requires that the original compiled data before packaging remain on disk where it was compiled for the Add and Update buttons to be available.
If you're uploading the mod to Steam Workshop for the first time, click on the 'Add to Workshop' button. You'll be guided through a series of prompts and questions to get the mod on the workshop. Once the mod is uploaded, you'll be presented with a link to view the mod on the Steam Workshop, and change any of its properties.
If you're updating a mod that you already published to the Steam Workshop, click on the 'Update Mod on Workshop' button. Again you'll be guided through a series of prompts and questions. When you are asked which mod you want to update, makes sure you pick the correct previously published mod. Once the mod is uploaded, you'll be presented with a link to view the mod on the Steam Workshop, and change any of its properties.
There are users that don't use Steam, so it's best to also upload the mod elsewhere for those users to install.

8. Examples

There's far too much complexity and detail to describe all the things you can change or add to the game (without spending months on documentation), so you'll have to learn by example with the game configuration data and trial and error. These few examples should get you started, but you'll want to get familiar with most things in the /resource folder.

A. A Simple Mod - Translation, Font, and Character Set

The first example, in /example/translation, shows a translation of all the strings in the game. It also changes the font and adds new characters to the font that are in use. Take a look in the /example/translation folder. It contains a mirror of some of the resources in the game, except they've been changed.

The example translation makes all text uppercase, and changes any vowels to have accents above them. Note that there are special characters in strings you'll want to keep intact - the help files have things like (^f0, ^jl, etc). Strings also have '~~' (for tilde) '~"' for quote, and @N (for substitution, where N is 0..9). During translation, you'll have to keep the escape characters intact.

First, you need to define the character set you want to use. You can view a character set example in /example/translation/Font/CharacterSet.rsc

```
// this file needs to be UTF16 little endian if it contains characters over index 255
// fonts that use this character set need to support all the characters listed here.
// This file is used to keep the font textures small, instead of having to include
thousands of characters
// ~ is a special escape character, so ~~ is required to add tilde to the set, ~" is
needed for quotation marks
CharacterSet resource
{
        String _characters = "
                // Special escape characters
                ~~ ~"
                // Punctuation
                `!@#$%^&*()_+-={}|[]\:;'<>?,./©
                // Numbers
                01234567890
                // characters for English (no lower case, note: help text will not
display correctly)
                ABCDEFGHIJKLMNOPQRSTUVWXYZ
                abcdefghijklmnopqrstuvwxyz
                // additional characters for this example
                ???
        ";
```

```
}
```

Next, you'll change the fonts for the game. There are four main fonts, each a different size. A font resource looks like this:

```
Font resource
{
        // reference character set
        CharacterSet _characterSet = "Font/CharacterSet.rsc";

        // whatever font you want, has to be loaded into windows
        String _fontName = "Arial";

        // height in pixels of the font
        int _fontHeight = 24;

        // name of the sprite sheet to create, and its size. You may have to increase
the size of
        // the sheet to contain all the characters
        String _sheetName = "Build/FontSheet.rsc";
        int _sheetWidth = 256;
        int _sheetHeight = 128;

        // name of the material that uses this font. The material needs to reference
        // the correct texture.
        String _materialName = "Font/FontMaterial.rsc";

        // the texture to output with all the glyphs on it.
        String _imageName = "Build/uiFontImage.png";
}
```

Finally, you can translate all the text. All text is contained in string tables that look like this:

```
StringTable common
{
        Entry _strings
        [
                { String _name = "Ok";                                  String _text
= "?"; }
                { String _name = "Cancel";                              String _text
= "C?C?"; }
                { String _name = "Yes";                                 String _text
= "Y?"; }
                { String _name = "No";                                  String _text
= "N?; }
                { String _name = "Apply";                               String _text
= "?PLY"; }
                { String _name = "Next";                                String _text
= "N?T"; }
                { String _name = "Quit";                                String _text
= "Q?T"; }
        ]
}
```

Note: If Unicode characters are required to support characters, resources files can be saved as USC-2, little endian. If you're unsure, the CharacterSet.rsc file is already in this format.

Depending on the translations and size of the text, some dialogs may not contain the text, or become misshapen. To remedy this, you could also modify parts of the user interface layout to accommodate the translation more fully.

For ease of building the mod, a single ExternalList resource is provided that references all the changed resources. To build this mod, run the command

```
bin\x64\Tools-x64.exe /build Package.rsc:list /pathres ../example/translation
```

```
/pathdat ../example/translation/bin
```

You should now be able to run the game with the command:

```
bin\x64\Application-x64-profile.exe /pathres ../example/translation /pathdat
../example/translation/bin
```

You'll see that the translation has taken effect. However, this has just modified the game data locally - this is great for quick development and iteration, but if you wanted to give this mod to others, you'd need to package it.

First, you'll create a definition for the package (see /example/translation/Package.rsc)

```
PackageFile translationExample
{
        String _name = "Translation Example";
        String _author = "Shining Rock Software, LLC";
        String _description = "This mod changes the font from the default to Arial.
                It also modifies the default character set. All text strings
                are made uppercase and vowels have accents over them.";
        String _icon = "icon.png";      // must be 48x48
        String _preview = "preview.jpg"  // preview image used for Steam Workshop,
must be square, png or jpg.
        int _userVersion = 1;

        // all files in resource directory (each item is searched recursively)
        String _includeList
        [
                "*"
        ]

        // exclude package files, mod files, file used for building packages
        String _excludeList
        [
                "Package_*.crs"
                "*.pkg"
                "*.pkm"
        ]
}
```

Next, you'll compile the package using the /mod parameter.

```
bin\x64\Tools-x64.exe /mod Package.rsc:translationExample /pathres
../example/translation /pathdat ../example/translation/bin
```

Building the mod package may take some time, depending on how many resources are in the data directory and how many have been modified. The tool checks the new resources against the ones contained in the game, and only packages files that have changed or are new. The tool will output files that it is packaging, so you can see what you've modified or added.
In the end, this will output a file called translationExample.pkm in the bin/WinData folder.

Next, test the mod properly.

1. When you start the game, start it so it only uses packages.

   ```
   bin\x64\Application-x64-profile.exe /onlypkg
   ```

   All the text should be as normal.
2. Open the Mod dialog by pressing the 'Mods' button.
3. The *Translation Example* mod should be listed in the dialog.
4. Enable it by checking the X button next to the text that says *Enable*.

5. Press OK.
6. The menu will prompt you to reload, and after doing so the mod will be in effect.
7. Once everything is working as intended in a mod, you can distribute the .pkm file to other users. All they have to do is place the .pkm file in the WinData folder, and then enable the mod in game.

B. A Building and Profession

This example mod adds a new building, a new edible item, and a new profession. In this case the building is an apiary, the building produces honey, and the profession is a bee keeper. If you want to build and test the mod...

```
bin\x64\Tools-x64.exe /build apiaryResources.rsc /pathres ../example/building
/pathdat ../example/building/bin
```

You should now be able to run the game and see that the apiary is available from the food menu. Note that before the mod is packaged, you have to explicitly tell the game to load the resources using the */ref* parameter. In the translation example, resources were simply overridden so no reference was required - the game references them already. Without the reference the new resources won't be loaded.

```
bin\x64\Application-x64-profile.exe /ref apiaryResources.rsc /pathres
../example/building /pathdat ../example/building/bin
```

Start a new game, and click on the Food Production icon - you should see the new building listed with an icon of honeycomb. You can place the building, have it built, and assign workers to it. In the professions dialog, the new profession of Bee keeper should be listed as well.

As with the translation example, you can build the mod as a package like so:

```
bin\x64\Tools-x64.exe /mod Package.rsc:apiary /pathres ../example/building
/pathdat ../example/building/bin
```

To test the mod by itself, you can run the game forcing it to only use packages.

```
bin\x64\Application-x64-profile.exe /onlypkg
```

The game should have the new mod listed in the Mods dialog and you can enable it.

Adding a new building requires many resources that are tied together - meshes, textures, materials, entity descriptions, text, sprites, and terrain decals. Below are the general steps to making a new building.

General steps to making a new building and profession.

1. Model the building and all intermediate construction models.
    - See /example/building/Models/apiary.fbx
2. Make sure the objects are properly centered around their pivot. For example, a building that is 4x5m and it's corner is at (0, 0, 0) should have the pivot at (2, 2.5, 0)
3. Assign materials to the building. The name of the material defines the material name on disk. Materials need to be in a MaterialInstance folder, in the same directory as the model. If not found there, the tool will check parent directories up until the resource root for a properly named material.
4. Add dummy helper objects to define locations for use.
    - Helpers named build_XXX are where workers will stand when constructing the building
    - Helpers named use_XXX are where workers will stand when working at the building.
    - Helpers named create_XXX are resources will be created at the building.
    - Helpers for particle system attachments can be named arbitrarily, for example, the smoke from the building.
5. Unwrap the model for texturing.
6. The first UV channel for a building is used by the material with the primary texture (unless you

make new shaders with different behavior).

7. The second UV channel is used for an ambient occlusion texture. Each construction model needs its own AO texture.
8. If the building has sections above the terrain, make a very low res 'floor' object that citizens will walk on. (The apiary doesn't need or use one.)
    - See /resource/Models/Buildings/FishermansDock/FishermansDockFloor.rsc
    - See /resource/Template/FishermansDock.rsc
9. Export the model in FBX format.
10. Build a material resource for the game to use.
    - See /example/building/Models/MaterialInstance/Apiary.rsc
11. Build a texture resource for the game to use.
    - See /example/building/Models/MaterialInstance/ApiaryTexture.rsc
12. Build AO texture resources.
    - See /example/building/Models/MaterialInstance/ApiaryAO.rsc
13. Make a model resource for the building and each step of construction. This is the visual display of the model. These files also contain an 'edge mesh' that allows the game to display the highlight around the visual mesh when selected.
    - See /example/building/Models/ApiaryMesh.rsc, ApiaryBuild01Mesh.rsc, ApiaryBuild02Mesh.rsc
14. Make a picking resource for the building and each step of construction. This is used for picking buildings with the mouse.
    - See /example/building/Models/ApiaryPicking.rsc, ApiaryPicking01Mesh.rsc, ApiaryPicking02Mesh.rsc
15. Make a point resource for the building that defines where workers will stand when creating, using, and building the building. This point file references the dummy helper points setup when modelling.
    - See /example/building/Models/ApiaryPoints.rsc
16. Make a floor resource if the building has parts that citizens walk on that are above the terrain.
17. Make a decal that will be placed on the terrain. If you're making multiple buildings you can put all the decals on the same texture. Putting more than one decal on a texture is better for rendering performance.
    - See /example/building/Models/MaterialInstance/ApiaryFootprint*
18. Make any sprites that are associated with the building. If you're making multiple buildings you can put all the sprites on the same sheet. Putting as many sprites in one sprite sheet is better for rendering performance.
    - See /example/building/UI/ApiarySpriteSheet*
19. Make any strings that are associated with the building. If you're making multiple buildings you can put all the strings in a single string table.
    - See /example/building/UI/ApiaryStringTable.rsc
20. Make a new profession if needed.
    - See /example/building/Profession/Profession.rsc
21. Make any new resources that may be needed by the building. Note that this type of resource has additional models, sprites, textures, and strings associated with it. You create them just like in the previous steps of creating a model of a building.
    - See /example/building/Template/RawMaterialHoney.rsc
    - See /example/building/Models/honey.fbx
    - See /example/building/Models/HoneyMesh.fbx
    - See /example/building/Models/MaterialInstance/Honey*
22. Setup an entity that ties together all the resources and defines how the building works. Setting up entities is complicated. For more examples, see all the game entities in /resource/Template/.
    - See /example/building/Template/Apiary.rsc
23. Add a toolbar item that adds the building to the main game toolbar.
    - See /example/building/Apiary.rsc
24. Make a resource reference file that will load all resources. This resource needs to be named <ModName>Resources.rsc:resource. If not, it will not load properly when packaged in mod format.

- See /example/building/apiaryResources.rsc

In many cases it's easier to iterate on models and textures while viewing them in game. If you build simple stub versions of the textures and models and get them into game - you can have the game reload them while you add details. Simply leave the game running and it will detect changes when you export fbx or pngs. Only Application-profile will detect changes. The final release executable does not. Note that this only works with loose resources. A mod already packaged into a single file will not reload resources, especially if the /onlypkg flag is used.

You can also change other resources, such as the entity definition, however some changes may make the game crash, especially if you change the number of items in an array. For example, changing the list of dummy node helpers and re-exporting while the game is running generally causes a crash. Other items, such as UI definitions, will reload, but won't take effect until the UI is closed and then reopened. Other values, such as storage amounts in buildings don't take effect until you build a new building.

For performance reasons it's best practice to have a building be a single object with one material. The game supports multiple objects with multiple materials, but it will take the game longer to rendering these objects.

C. Debug Options

For testing, it's useful to enable debug mode so you can place buildings without needing resources. You can also add resources, citizens, change time of year, change the color of fog and lighting, cause disasters, and see general performance statistics.

To enable it, you can edit

```
/resource/Game/Toolbar.rsc
```

and find the 'debug' Toolbar item. Change the

```
bool _visible = false;
```

to

```
bool _visible = true;
```

Next time you load the Tools & Reports toolbar, there will be a debug option, which will bring up a dialog with many debugging tools. Note that the game isn't guaranteed to work properly with debug/cheating enabled. You can break the game by changing the time slider for seasons, and invalidate some state that may cause crashes.

If you're building a mod in a separate directory from the normal resource folder, you'll have to copy the Toolbar resource into your directory in the same relative place so that the game will see the change.

D. Adding a new crop

This example adds a new crop to fields. In this case the crop is lettuce. If you want to build and test the mod...

```
bin\x64\Tools-x64.exe /build lettuceResources.rsc /pathres ../example/crop
/pathdat ../example/crop/bin
```

You should now be able to run the game and see that lettuce can now be used when you plant crops in a field. You'll most likely want to enable debug mode to be able to use the lettuce seeds before buying them from a trading post. You'll have to copy the toolbar resource (from the previous example) into the /example/crop/Game folder so that you can enable debug mode. The debug menu is available from the

Tools & Reports menu item. There's an icon of a bug on it. When you bring up the debug dialog, there's a button to give all seeds to the player. Just hover over the buttons and read the tooltips.

Before giving a mod to others, make sure you've turned the debug option off if it isn't your intent to leave it enabled.

Before the mod is packaged, you have to explicitly tell the game to load the resources using the */ref* parameter.

```
bin\x64\Application-x64-profile.exe /ref lettuceResources.rsc /pathres
../example/crop /pathdat ../example/crop/bin
```

Start a new game, and use the debug menu to give all seed types to the player. Then place a crop field - you should be able to select Lettuce as the crop. This crop works like any other.

You can build the mod into a package like so:

```
bin\x64\Tools-x64.exe /mod Package.rsc:lettuce /pathres ../example/crop /pathdat
../example/crop/bin
```

To test the mod by itself, you can run the game forcing it to only use packages.

```
bin\x64\Application-x64-profile.exe /onlypkg
```

The game should have the new mod listed in the Mods dialog and you can enable it.

If you've been working through all these examples, you can see you can load multiple mods at once. You can start a game with both the lettuce mod and the apiary mod.

General steps for adding a new crop

1. You'll need to model two objects. One for this display of the crop in the field, and one for the crop when it is harvested.
    - See /example/crop/Models/Lettuce.fbx
2. Like in the building example you need to build Textures, Materials, and Meshes for the objects. The process is the same.
    - See /example/crop/Models/MaterialInstance/Lettuce* and LettuceLeaf*
    - See /example/crop/Models/LettuceLeafMesh.rsc and LettuceLeaf.rsc
3. Like in the building example you need to build String Tables and Sprites.
    - See /example/crop/UI/CropSpriteSheet.rsc
    - See /example/crop/UI/CropStringTable.rsc
4. Finally you'll have to make entity definitions (again, similar to the building example)
    - See /example/crop/Template/NaturalResourceLettuce.rsc
    - See /example/crop/Template/RawMaterialLettuce.rsc
5. To package and reference the new entities, you'll need a package resource and a list of referenced objects.
    - See /example/crop/lettuceResources.rsc
    - See /example/crop/Package.rsc

E. Adding a new tree

This example adds a new tree to orchards. In this case the tree produces figs. If you want to build and test the mod...

```
bin\x64\Tools-x64.exe /build figResources.rsc /pathres ../example/tree /pathdat
../example/tree/bin
```

You should now be able to run the game and see that fig trees can now be used when you plant trees in a

orchard. Like the crop example, you'll need to enable debug mode in this mod to be able to give yourself fig seeds without buying them at a trading post.

Before the mod is packaged, you have to explicitly tell the game to load the resources using the */ref* parameter.

```
bin\x64\Application-x64-profile.exe /ref figResources.rsc /pathres ../example/tree
/pathdat ../example/tree/bin
```

Start a new game, and use the debug menu to give all tree types to the player. Then place a orchard - you should be able to select Figs as the fruit. This tree works like any other.

You can build the mod into a package like so:

```
bin\x64\Tools-x64.exe /mod Package.rsc:fig /pathres ../example/tree /pathdat
../example/tree/bin
```

To test the mod by itself, you can run the game forcing it to only use packages.

```
bin\x64\Application-x64-profile.exe /onlypkg
```

The game should have the new mod listed in the Mods dialog and you can enable it.

General steps for adding a new tree

Adding a new tree is almost identical to adding a new crop, just the entity definitions are different, and modeling the tree can be slightly challenging.

1. You'll need to model two objects. One for this display of the tree in the field, and one for the fruit when it is harvested. There is a slight difference is the modelling of the tree.

   Deciduous trees in the game use the a vertex shader to always make the leaf billboards point at the camera. When modelling the leaves only a small quad (0.01m) is needed for each billboard. The billboards have a third texture coordinate that tells each vertex how far to expand and in what direction.

   In general the billboards are also mapped to a location on the Ambient Occlusion texture that is fully white - and actual AO information is only used on the tree trunk.

   On other parts of the tree, such as the trunk, make sure the third texture coordinate is mapped to uv (0, 0), otherwise the trunk will sway in the wind along with the leaves. If you're adding a new conifer style tree, the third texture coordinate just defines how much the leaves sway in the wind.
   - See /example/tree/Models/FigTree.fbx
2. Like in the crop example you need to build Textures, Materials, and Meshes for the objects. The process is the same.
   - See /example/tree/Models/MaterialInstance/Fig* and FigTree*
   - See /example/tree/Models/FigMesh.rsc and FigTreeMesh.rsc
3. Like in the crop example you need to build String Tables and Sprites.
   - See /example/tree/UI/FigSpriteSheet.rsc
   - See /example/tree/UI/FigStringTable.rsc
4. Finally you'll have to make entity definitions (again, similar to the building example)
   - See /example/tree/Template/NaturalResourceFig.rsc
   - See /example/tree/Template/RawMaterialFig.rsc
5. To package and reference the new entities, you'll need a package resource and a list of referenced objects.
   - See /example/tree/figResources.rsc
   - See /example/tree/Package.rsc

F. Adding a new animal

This example adds a new animal to pastures. In this case the animal is a white chicken. It behaves just like the other chickens in the game. If you want to build and test the mod...

```
bin\x64\Tools-x64.exe /build whiteChickenResources.rsc /pathres ../example/animal
/pathdat ../example/animal/bin
```

You should now be able to run the game and see that 'Leghorns' can now be used when you select an animal in a pasture. 'White Chicken' does fit on all the UI very well, so 'Leghorns' which is a type of white chicken, is used instead. This is something to keep in mind as you mod the game - make sure your text fits in the UI. The UI will expand controls to fit the text, however this may cause the entire UI to stretch in unexpected ways.

Like the crop example, you'll need to enable debug mode in this mod to be able to give yourself all the animal types without buying them at a trading post.

Before the mod is packaged, you have to explicitly tell the game to load the resources using the */ref* parameter.

```
bin\x64\Application-x64-profile.exe /ref whiteChickenResources.rsc /pathres
../example/animal /pathdat ../example/animal/bin
```

Start a new game, and use the debug menu to give all animal types to the player. Then place a pasture - you should be able to select 'Leghorns' as the animal. This animal works like any other. There's another debug button to fill pastures with animals without waiting for them to grow. Press it a few times to add the animals.

You can build the mod into a package like so:

```
bin\x64\Tools-x64.exe /mod Package.rsc:whiteChicken /pathres ../example/animal
/pathdat ../example/animal/bin
```

To test the mod by itself, you can run the game forcing it to only use packages.

```
bin\x64\Application-x64-profile.exe /onlypkg
```

The game should have the new mod listed in the Mods dialog and you can enable it.

General steps for adding a new animal

Adding a new animal is almost identical to adding a new crop or tree, just the entity definitions are different, and the model requries bones, skinning, and animations.

1. You'll need to model two or three objects. One for this display of the animal in the field, and one for the meat when it is harvested. You may also need a model for another produced resource - chickens produce eggs, and cows produce milk.

   After modelling the animal, you'll need to rig it with bones and weight the mesh to the bones using a skin modifier. You can setup any sort of animation controllers, ik controllers, etc, as long as when you export the FBX file you bake the animation into the file, so that all animation frames are explicit, instead of relying on the controllers.

   It's important to keep the number of bones low. A maximum of 51 bones is supported, however for the game to render many animals quickly (in DirectX 9 mode), you'll want to keep the bone count much lower than that. The lower the bone count, the more models the game can render simultaneously.
   - See /example/animal/Models/WhiteChicken.fbx

2. Like in the crop example you need to build Textures, Materials, and Meshes for the objects. The process is the same.
    - See /example/animal/Models/MaterialInstance/WhiteChicken*
    - See /example/animal/Models/WhiteChickenMesh.rsc
3. You'll need to define an animation file that tells the game the ranges of animation for various motions. Animals need a walk, and three idles. It's important to make sure animations that are played often on many of the same object be properly instanced, and that the number of different instances running is fairly low.
    - See /example/animal/Models/WhiteChickenAnims.rsc
4. Like in the crop example you need to build String Tables and Sprites.
    - See /example/animal/UI/WhiteChickenSpriteSheet.rsc
    - See /example/animal/UI/WhiteChickenStringTable.rsc
5. Finally you'll have to make entity definitions (again, similar to the all the other examples, just different configuration). The white chicken reuses the meat and eggs resources that already exist in the game, so there aren't new resources for them.
    - See /example/animal/Template/WhiteChicken.rsc
6. To package and reference the new entities, you'll need a package resource and a list of referenced objects.
    - See /example/animal/whiteChickenResources.rsc
    - See /example/animal/Package.rsc

G. Using Custom Resource Types and Limits

This example shows how to use custom resource types and limits. Previous versions of the game were limited to the built in resource types. Food, Iron, Wood, etc. There are 10 Custom resource limits that can be used however modders see fit. Once the mod is loaded, the status bar, resource limit window, and the town hall production and graphs will show an additional set of resources. Currently these are simple called Custom0, Custom1 ... Custom9. It's up to a modder to change the string tables to define what each custom resource is. To build this mod, run

```
bin\x64\Tools-x64.exe /build limitResources.rsc /pathres ../example/limitresource
/pathdat ../example/limitresource/bin
```

You can build the mod into a package like so:

```
bin\x64\Tools-x64.exe /mod Package.rsc:limitResource /pathres
../example/limitresource /pathdat ../example/limitresource/bin
```

To test the mod by itself, you can run the game forcing it to only use packages.

```
bin\x64\Application-x64-profile.exe /onlypkg
```

There are 10 custom resources available. This mod doesn't actually modify any buildings or resources to use the custom types. Here's an overview on how you would change the Apiary to use a custom resource type and limit.

1. In the building example, open Template/Apiary.rsc.
2. Modify the ConsumeProduceDescription to use a custom resource type. It currently is food. Change

    ```
    ResourceLimit _resourceLimit = Food;
    ```

    to

    ```
    ResourceLimit _resourceLimit = Custom5;
    ```

3. Modify the ResourceLimitUIConfig to use a custom resource type. Again change

```
ResourceLimit _resourceLimit = Food;
```

to

```
ResourceLimit _resourceLimit = Custom5; // this could be Custom[0..9]
```

4. Modify the tooltip UI elements to change the text on the tooltip

```
String _toolTipText = "FoodLimitTip";
```

becomes

```
String _toolTipText = "Custom5LimitTip";
```

5. Modify the text UI element to change the text on the dialog

```
String _text = "FoodLimit";
```

becomes

```
String _text = "Custom5Limit";
```

6. Note that in this mod Template/Citizen.rsc and Template/StorageBarn.rsc have been modified to be able to store certain resource types. For your own storage areas you'll have to modify them as needed.

```
RawMaterialFlags _storageFlags = Edible | Fuel | Tool | Wood | Stone | Iron |
Health | Clothing | Textile | Alcohol |
                   Custom0 | Custom1 | Custom2 | Custom3 | Custom4 | Custom5 |
Custom6 | Custom7 | Custom8 | Custom9;
```

7. If desired change all the string table entries for Custom5 to match whatever you want your resource type and limit to be called. There are several places to change in the StringTable.rsc. Their names are
    - Custom5Limit
    - Custom5LimitShort
    - Custom5LimitTip
    - Custom5Tip
    - Type18 (this string entry is the named use when displayed as a graph in the town hall)

   You'll want to eventually modify the user interfaces to only display the limits and graphs that you want. Generally look for the word 'custom' in these resource files:
    - Dialog/StatusBar.rsc contains the custom items listed on the Status Bar in game. Any custom field can be ommited if not all 10 custom resource limits are used.
    - Dialog/TownHall.rsc contains fields where resource limits are placed used on the Production Tab in the town hall. It also modifies the production tab to have a scroll bar to keep the town hall window small. Some additional UI resizing may be needed.
    - Template/TownHall.rsc contains the UI config that adds the resource limits to the TownHall dialog.
    - Template/UtilityLimits.rsc contains the UI config that add the resource limits to the limits dialog.
    - Dialog/StringTable.rsc contains strings used to name the custom resources. To remove custom resources that are unused from the graphs, simply leave the strings blank, or truncate the list.
    - Template/TradingPost.rsc contains changes to add custom resource types to the auto purchase UI list. Note that if you change this list you need to update the number of entries in the table in the UI. This is located in Dialog/Trade.rsc. (TableDescription tablePurchase needs the _height value to match the number of entries)

   Note that if two mods are loaded that both use a custom resource types that clash, behavior is

undefined - resources you expect to be grouped with a certain type may have the wrong name and logically unrelated types will end up with the same resource type and limit.

H. More than 3 resource requirements for a building

This example shows how to use more than 3 resource items when constructing a building. Previous versions of the game were limited to 3, and if more were used a crash was seen. Once the mod is loaded, the basic wooden house requires six items for construction, Wood, Stone, Iron, Tools, Firewood, and Coal To build this mod, run

```
bin\x64\Tools-x64.exe /build buildRequirement.rsc /pathres
../example/buildrequirement /pathdat ../example/buildrequirement/bin
```

You can build the mod into a package like so:

```
bin\x64\Tools-x64.exe /mod Package.rsc:buildRequirement /pathres
../example/buildrequirement /pathdat ../example/buildrequirement/bin
```

To test the mod by itself, you can run the game forcing it to only use packages.

```
bin\x64\Application-x64-profile.exe /onlypkg
```

Here's a basic list of changes required to use more resources when constructing buildings

1. In ToolTipToolBar.rsc, change the layout to have as many labelBuildX as are required (i.e. labelBuild3, labelBuild4, ...)
2. In CreateBar.rsc, change make a similar change adding more labelBuildX as are required
3. In Build.rsc change the storageTable to have as many entries as are required, plus an additional one thats used for the amount of work required.
4. In whichever building you want, change the build section to have more items, See WoodHouse.rsc:build.
5. Optional: If desired, per building, the UI can be changed to match the number of required resources. You'd have to change the UIDescription to point at a dialog with the correct number of items in the storage table, instead of the default "Dialog/Build.rsc:build"

9. Change List

Changes for Banished build 170910

- - Debug menu now supports adding custom resources.
- - Crashes fixed when trying to use more than 3 resource requirements on a building.
- - Fixed a crash caused by using UI scale in some cases.
- - Updated the mod kit to have an example of modifying the trading post to auto purchase custom resources.
- - Added Custom0..9 resource flags and resource limits. Modified UI that uses these accordingly to be able to use them.
- - Fixed a bug cause by towm hall deletion while nomads are in route

Changes for Banished build 160521

- - Fixed a crash that occurred when clicking on the town hall if a translation mod was in use that was built with 1.0.4. Missing text data will now be blank.
- - Fixed a bug that caused orchards and pastures to not drop items inside their boundaries as was intended.

Changes for Banished build 151214

- - Fixed a bug that caused fonts from 1.0.4 to not load in 1.0.5. A UCS2 - UTF8 conversion wasn't made properly.
- - Fixed a bug that caused dropped resources (from citizen death/task cancelation) to drop in invalid places.
- - Fixed a bug that caused orchards to cause invalid data access and or data corruption if a citizen tried to harvest a tree, but the tree died before he got there.
- - Fixed a bug that caused potential memory corruption when cutting down an orchards trees.
- - Fixed a bug that caused a crash if game startup failed before memory allocation was available or was corrupt. It now properly displays an error.
- - Fixed a bug that caused a crash when loading old mods that had custom materials. The game will no longer crash, however objects with those materials will not display. Mods should be updated to the newest mod kit version to fix this issue.
- - Added better error message if the game runs out of memory due to too many mods loaded.

Changes for Banished build 151026

- - UTF8 is now used instead of USC2.
- - Resource files can be in UTF8, USC2, UTF16, big and little endian. They'll be converted to UTF8 on load.
- - Memory usage allowance has been increased to 1 gigabyte, which should allow for larger mods.
- - All materials now use custom shading language SRSL instead of HLSL.
- - Any mods with custom materials will need to be modified to point to the new shaders and/or use SRSL.
- - Math library can now be compiled without the need for SIMD instructions.
- - OpenGL is now supported (but isn't currently being released with the PC version)
- - Data compilation is now in a separate DLL - CompileWin.dll - this can be swapped out for other platforms (consoles, mac, linux, etc)
- - Shader compiler is now in it's own DLL. Video DX9/DX11/GL dlls are no longer required for compiling shaders.
- - Added safety code to check for invalid and dangling pointers - this should make catching hard to find and rare issues easier.
- - Sped up mod details dialog for massive mods that have 10000's of files included. This should make looking at conflicts and uploading to Steam workshop easier.
- - Beta Mods and Mods newer than the currently released version can no longer be uploaded to Steam Workshop.
- - Nvidia and AMD GPUs in laptops should now be auto selected for use, instead of an Intel Integrated card.
- - Textile limit is now available for modders to use.
    - - Cropfields, Fishing, Forester, Hunters, Orchards, and Pastures now have a configurable resource limit.
    - - Livestock has a resource limit for the by product they make (eggs, wool, milk, etc) Note that if a by product isn't created because of the resource limit, the icon won't appear above the building.
    - - Added textile to the Status Bar, Resource Limit window, and Town Hall UI
    - - Added graphs for textiles to Town Hall UI

Changes for Banished build 141123

- - Uploading mods to Steam Workshop now requires the user to have the original compiled data before packaging on disk (generally .crs files). The data must be in the same location it was build and match the files in the package. For example, if the mod was built in C:\BanishedKit\mymod\bin\, then all files created during mod compilation must remain in that directory for the mod to be added to Steam Workshop.
  Without the original data, the Add to Workshop and Update on Workshop buttons are unavailable. Current mods need to be rebuilt with Banished Kit 141123 before they can be updated.

No other changes to the game or mod kit have been made, so non-steam users can continue using build 141103.

Changes for Banished build 141103

- - Increased memory usage allowed for save games. This allows larger modded maps than default to be saved safely. However at some point very large maps will crash the game due to out of memory, or textures failing to be created.
- - Trade UI now expands automatically for orders.
- - Fix a bug that caused mods to become unreferenced in save games when opening the mod dialog in game and then pressing cancel.
- - Fixed a bug that caused mods that should be unloaded to stay in memory if they were loaded at game start. This fixed random ghost buttons on the toolbar.
- - Fixed tutorials not progressing properly.

Changes for Banished build 141003

- - Fixed mods not showing up on the toolbar.
- - Fixed crash when loading save games saved with 141001.

Changes for Banished build 141001

- - Fixed a crash when disabling mods that had new map types.
- - Fixed the toolbar showing empty buttons in certain cases.
- - The trade UI now dynamically resizes to make sure all items can be shown at once. This fixes crashes when the general merchant comes with many mods that add resources enabled.
- - Fixed a rare crash that was caused by closing tooltips in a particular way.
- - xWMAEncode now uses the /bin parameter to located it.
- - Fixed a bug that allowed combo box drop downs to stay up if the parent button went away.
- - Added an option to draw the mouse cursor in software in cases where DirectX fails to display the cursor properly.
- - Moved 'clip mouse to window' option from Game options to input options.
- - Added a cursor option that allows the resource to specify the hot spot of cursors. This also facilitates the software cursor.
- - Added a Reset All option to the game launcher. This allows the game to reset to default settings if a setting is causing a game crash. The command line option /reset also does the same thing. Note this will unset achievements in non-steam versions.
- - Adding an option to the game launcher to disable use of DirectInput. This should allow systems where the mouse doesn't move to work properly, however only the left, right, and middle mouse buttons will be available.
- - Roads will now always draw roads no matter how many there are on the terrain.
- - Misc fixes for rare crashes.

Changes for Banished build 140925

- - Updated to latest FBX sdk. This required moving x32 and x64 builds into their own folder since the dll name is the same for both 32 and 64 bit versions.
- - Added support for a cmd.txt file that contains command line parameters. cmd.txt must be in the same folder as the executable.
- - Added command line parameter /bin that tells the game where the main /bin folder is located relative to the executable. This allows a single bin/WinData folder to be used along side the x32 and x64 versions of the game.
- - Added a new variable to PackageFile resources, 'String _preview = "pngOrJpgPath";' This is a square preview image that is used when uploading mods to steam workshop. If not set, no preview image will be uploaded to steam.
- - Added Steam Workshop support. Mods can be created, updated, downloaded, and browsed in

game. Any updates to subscribed mods will download automatically. A reload is required once downloads are complete.

- - Fixed tooltip text going outside the background boundaries.
- - Fixed combo boxes not displaying correcly after being enabled, disabled, then re-enabled.

Changes for Banished build 140825

- Initial Version