

EightBitz's Dungeondraft Scripts

Contents

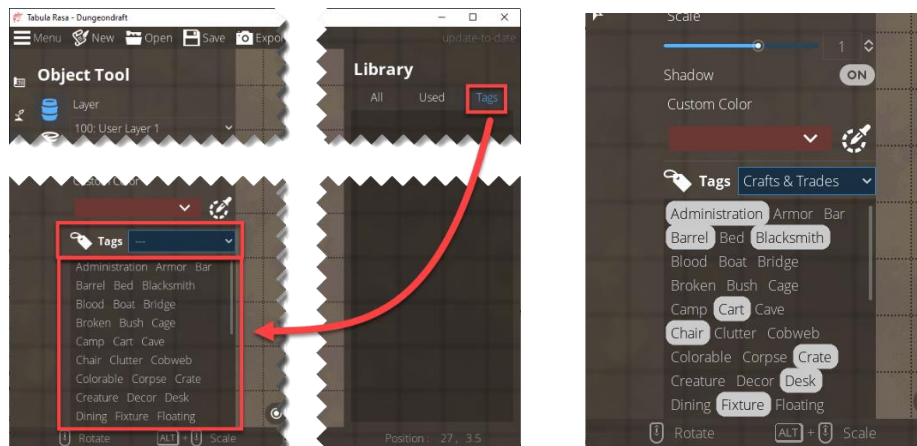
What Are These Scripts and Why Should I Use Them?.....	2
DDTagAssets.ps1	2
DDCopyAssets.ps1	3
DDConvertAssets.ps1	4
DDConvertPacks.ps1	4
dungeondraft-unpack.exe	4
dungeondraft-pack.exe	5
How Do I Copy these Scripts to My Computer?.....	5
Install PowerShell ISE	5
Create a Folder for the Scripts	5
Copy and Paste Each Script into PowerShell ISE	5
Download dungeondraft-unpack.exe and dungeondraft-pack.exe	9
Install ImageMagick	9
How Do I Use These Scripts?.....	10
Set the PowerShell Execution Policy	10
General PowerShell Syntax	12
Windows PowerShell vs. Windows PowerShell ISE	13
DDTagAssets.ps1	13
Description	13
Parameters	14
DDCopyAssets.ps1	15
Description	15
Parameters	16
DDConvertAssets.ps1	17
Description	17
Parameters	17
DDConvertPacks.ps1	17
Description	17
Parameters	18

What Are These Scripts and Why Should I Use Them?

[DDTagAssets.ps1](#)

The DDTagAssets.ps1 script is probably the most useful one of these. It's the first one I wrote, and it's kind of the centerpiece of the collection. This script is for people who want to make custom asset packs, either for their own use or to distribute to the community. But why? What does it do?

When you open Dungeondraft, you will notice that you can filter objects by tags and tag sets.



Clicking on any of the tags in the list will show you objects related to those categories.

Clicking on the dropdown arrow will show you a list of broader categories, each of which will show you objects related to a set of categories.

If you want the assets in your pack to be sorted by tags and tag sets, you have to create a file that defines which objects are associated with which tags and which tag sets. An example file would look like this:

```
{
  "tags": {
    "MyTag": [
      "textures/objects/sample_barrel.png",
      "textures/objects/sample_cauldron.png"
    ],
    "Colorable": [
      "textures/objects/sample_cauldron.png"
    ]
  },
  "sets": {
    "Example Set": [
      "MyTag"
    ]
  }
}
```

You would have to create a file in that format to include each object under its proper tag. You would have to make sure the file is properly formatted, otherwise Dungeondraft is going to crash when you try to load the asset pack.

If you just want to assemble one or two small asset packs, that might not be an issue. But the more asset packs you create, and the larger they get, the more tedious it gets to keep creating these tag files manually.

You can exclude them completely, but then the objects in you pack will be completely unsorted.

So those are your two options. Create tag files manually, which can get tedious and frustrating, or ignore tag files completely, which will keep your objects unsorted.

Alternatively, you can use an automated process, like this script, that makes it quick, easy and simple to create tag files for your custom asset packs.

[DDCopyAssets.ps1](#)

The DDCopyAssets.ps1 script will copy a folder structure containing supported image types to a folder structure in preparation for packaging by Dungeondraft.

Supported image types include: bmp, dds, exr, hdr, jpg, jpeg, png, tga, svg, svgz and webp.

Normally, this would be a straightforward drag-and-drop process that wouldn't require a script, but I wrote this script to copy assets from Campaign Cartographer 3 Plus (abbreviated as CC3+).

If you use this script on non-CC3+ folders, it will still copy the files, but for CC3+ folders, it does some extra work.

CC3+ folders contain duplicate images at various levels of quality: very high, high, standard, low and very low. For example if there's an image named "House1", it's CC3+ folder would contain five different versions of that image:

```
House1_VH.PNG  
House1_HI.PNG  
House1.PNG  
House1_LO.PNG  
House1_VL.PNG
```

Not all files in CC3+ are available in all five versions. Some files will only be available as *_HI.PNG, and some only as *.PNG. This script looks for all versions, then picks whichever one is the highest quality and copies only that version. In the example above, it would only copy House1_VH.PNG.

Also, CC3+ designates doors and windows by starting the file names with "Door" or "Window" followed by a trailing space ("Door " or "Window "). This script will optionally copy such files to the portals folder instead of the objects folder, allowing Dungeondraft to recognize these images as portals.

DDConvertAssets.ps1

The DDConvertAssets.ps1 script will convert images of all supported types (bmp, dds, exr, hdr, jpg, jpeg, png, tga, svg, svgz) to webp.

When Dungeondraft loads asset packs, it loads them into memory. The larger your asset packs are, and the more of them you load, the more your performance will be affected, and there's a threshold where Dungeondraft might crash.

Webp is a compressed format with low-loss.

With one of my folders, the .PNG version is 796 KB while the .WEBP version is 422 KB.

With another folder, the .PNG version is 221 MB while the WEBP version is 27 MB.

I don't know how WEBP compares to other formats, but with PNG files, at least, converting to WEBP can improve your performance while allowing you to load more assets and prevent crashes.

This script will also find and update any data files to change all filename references to have the .webp extension.

This script uses ImageMagick to do the conversion. If ImageMagick is not installed, this script will fail.

ImageMagick can be downloaded from <https://imagemagick.org/index.php>

DDConvertPacks.ps1

The DDConvertPacks.ps1 script converts .dungeondraft_pack files to WEBP.

This script uses dungeondraft-unpack.exe extract the pack files into folders.

Then it uses DDConvertAssets.ps1 to convert the image files to webp.

Then it uses dungeondraft-pack.exe to package the converted folders into pack files.

By default, this script creates and removes intermediate working folders, but you can specify an option to leave the working folders in place instead of removing them.

Dungeondraft-unpack.exe and dungeondraft-pack.exe were written by Ryex (Ryex#5366 on Megasploot's Discord server), and they can be downloaded from <https://github.com/Ryex/Dungeondraft-GoPackager/releases/tag/v1.0.0>

dungeondraft-unpack.exe

This program unpacks .dungeondraft_pack files. It was written by Ryex (Ryex#5366 on Megasploot's Discord server), and it can be downloaded from <https://github.com/Ryex/Dungeondraft-GoPackager/releases/tag/v1.0.0>

dungeondraft-pack.exe

This program creates .dungeondraft_pack files from existing folders. It was written by Ryex (Ryex#5366 on Megasploot's Discord server), and it can be downloaded from

<https://github.com/Ryex/Dungeondraft-GoPackager/releases/tag/v1.0.0>

This program will not create the requisite pack.json file that Dungeondraft requires. In order for this program to create a usable .dungeondraft_pack file, the folder will have had to have been packed by Dungeondraft at some point in the past so it contains a valid pack.json file.

In other words, if you're packaging a folder for the first time, you have to do it with Dungeondraft, not with this program.

How Do I Copy these Scripts to My Computer?

Install PowerShell ISE

I would first recommend installing PowerShell ISE. If you're on Windows 10, you may already have it installed, but if you go through this document and discover that it's not installed, you can find directions for installing it at <https://winaero.com/blog/install-or-uninstall-windows-powershell-ise-in-windows-10/>

Create a Folder for the Scripts

Create a "DDScripts" folder somewhere convenient. Suggestions could be:

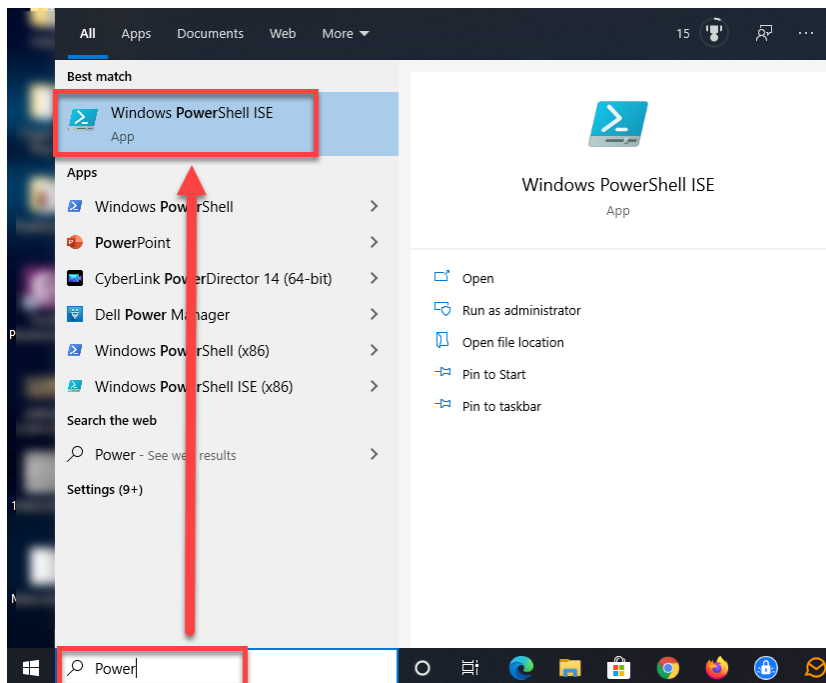
C:\Users\<your username>\scripts\DDScripts

C:\Users\<your username>\Documents\Dungeondraft\Custom Assets\DDScripts

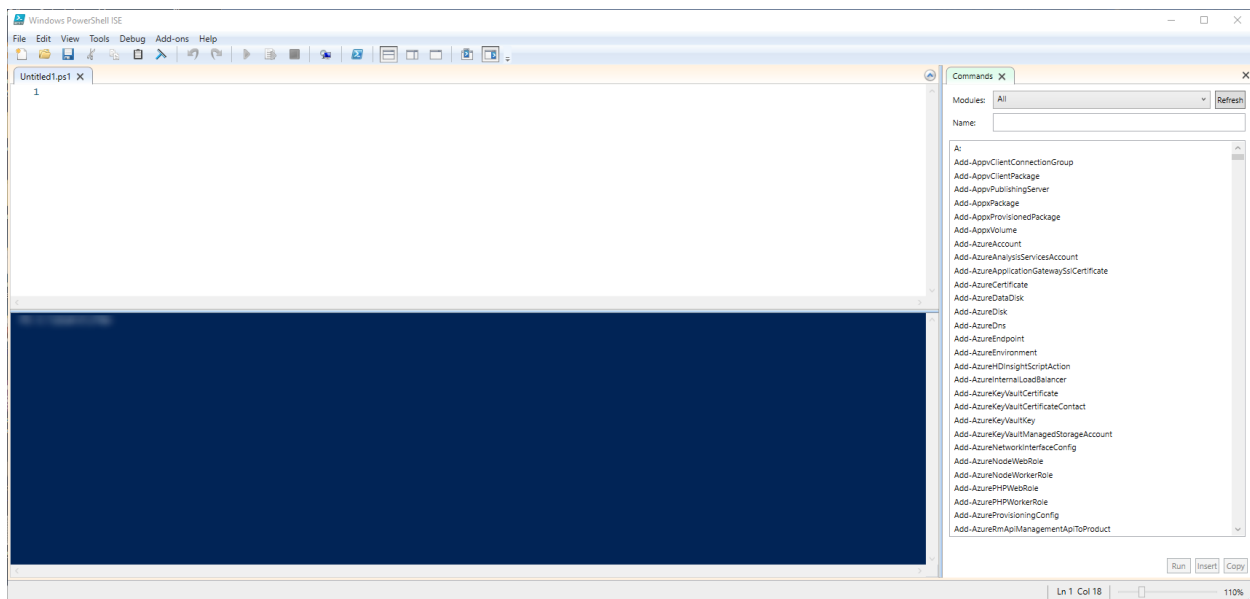
For the rest of this document, I'm going to assume that wherever you create the folder, you're naming it "DDScripts". When I refer to "DDScripts" or "the DDScripts folder", I'm referring to this folder.

Copy and Paste Each Script into PowerShell ISE

Open Windows Powershell ISE by typing the name in the Windows search box.

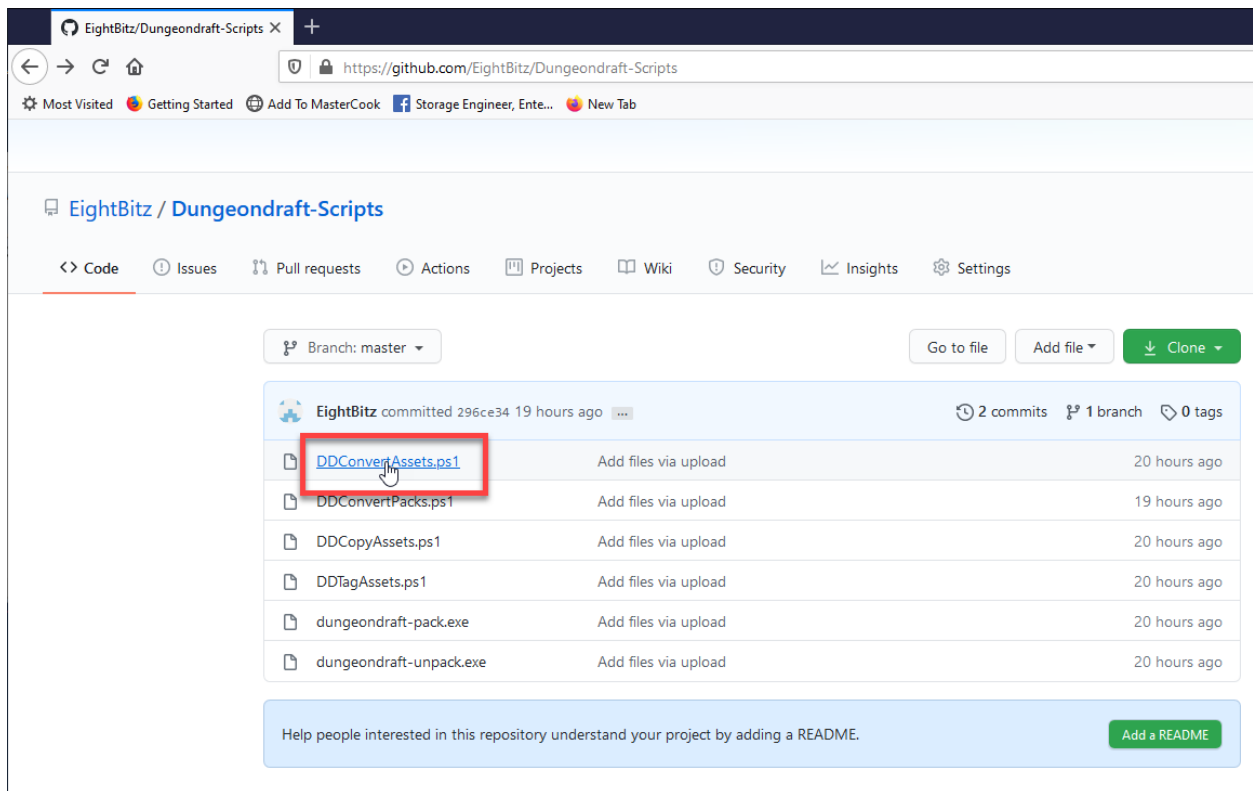


That should open a window that looks like this:

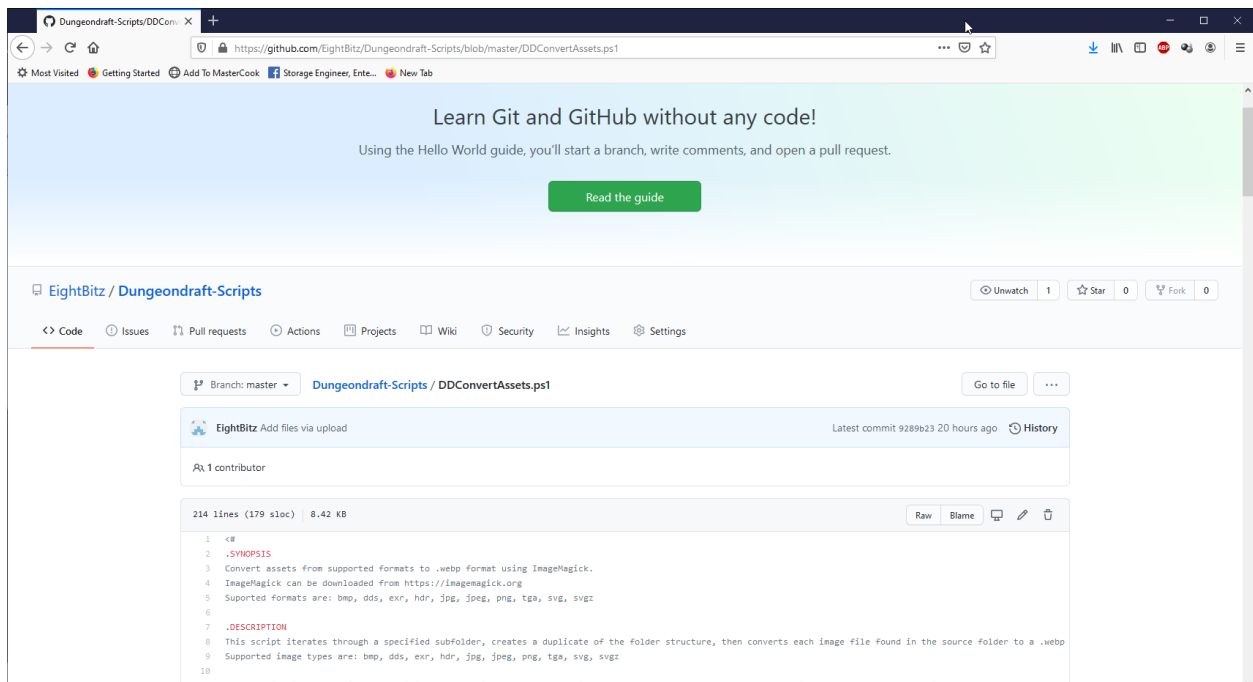


Browse to <https://github.com/EightBitz/Dungeondraft-Scripts>

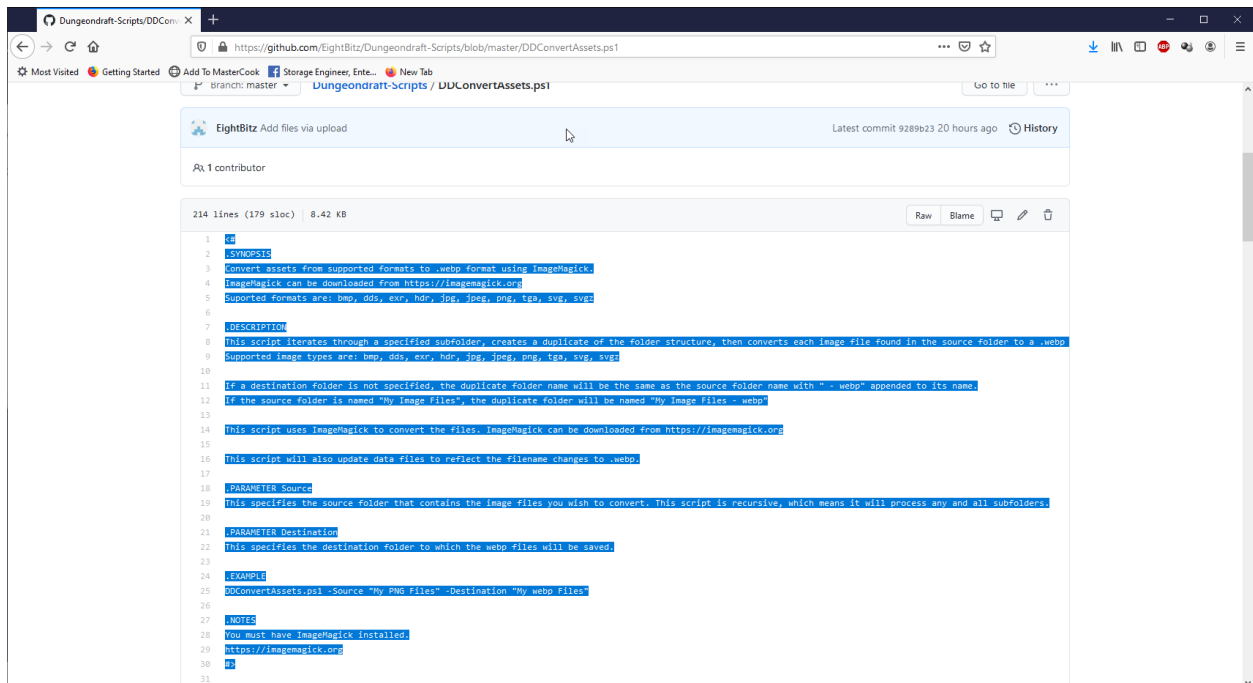
Your browser window should show you the list of scripts. Click on the first script in the list, DDConvertAssets.ps1.



That will take you to a page that shows the full script.

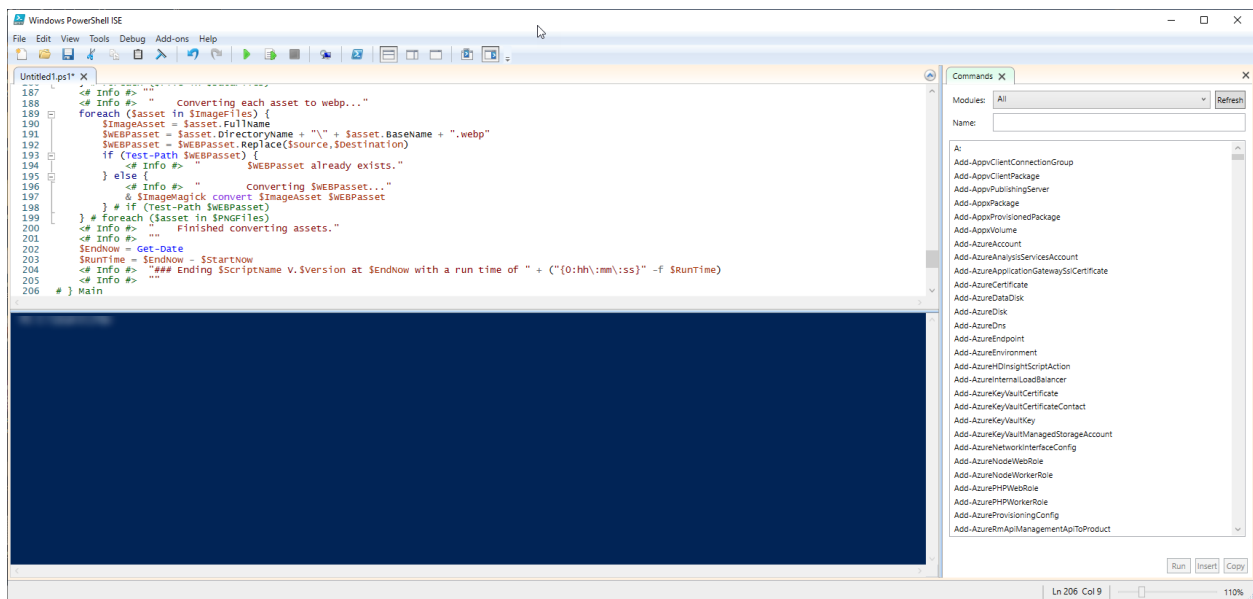


Highlight the full text of the script window (not the whole web page, just the script window). Unfortunately, you can't do this with CTRL-A to select all, because that will select the whole web page. You just want the script. You will have to hold the left mouse button to highlight while scrolling.



After highlighting the entire script, copy it to the clipboard (by pressing CTRL-C, or by right-clicking the selected text and selecting “Copy”).

Paste the script into the top half of the PowerShell ISE window.



To make sure you have the entire script, the first two lines should be:

```
<#
.SYNOPSIS
```

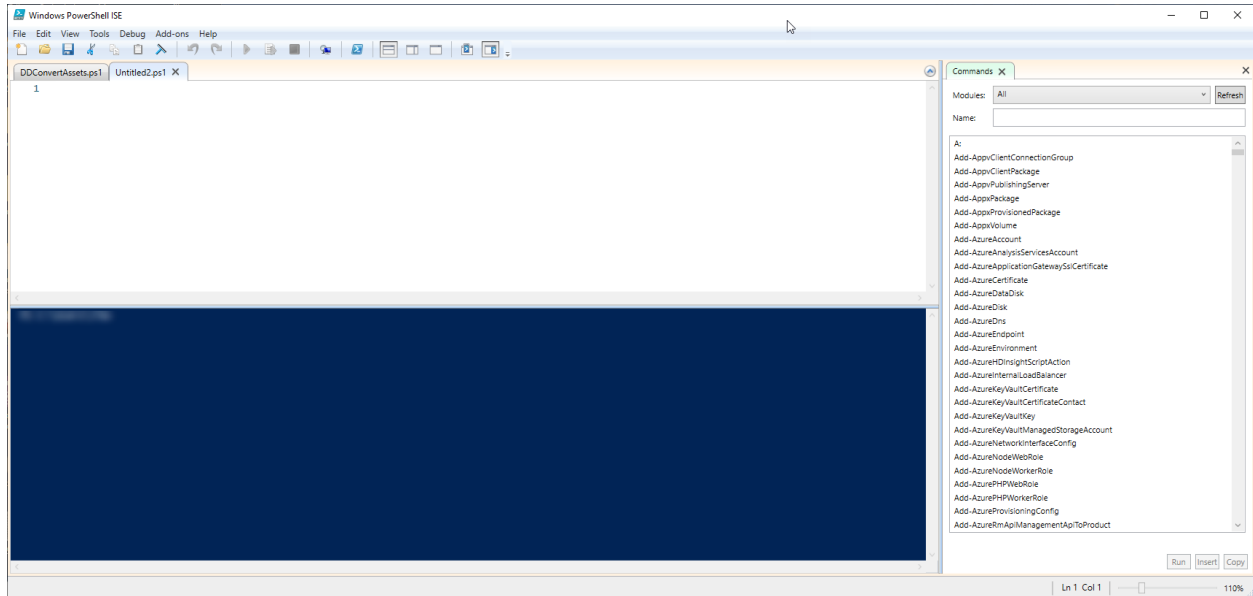
And the last line should be:

```
# } Main
```


In the PowerShell ISE window, click the File menu, and select “Save.”

Save the file to your DDScripts folder with a name of “DDConvertAssets.ps1”

In the PowerShell ISE window, click the File menu, and select “New.”



In the same manner as above, copy and paste each of the other .ps1 scripts into a new file, and save each one to your “DDScripts” folder with its correspond name: “DDConvertPacks.ps1”, “DDCopyAssets.ps1” and “DDTagAssets.ps1”.

As you copy and paste each script, verify you’ve pasted the entire script by verifying the same, two first lines for each:

```
<#  
.SYNOPSIS
```

And the same last line for each:

```
# } Main
```

Download dungeondraft-unpack.exe and dungeondraft-pack.exe

Browse to <https://github.com/Ryex/Dungeondraft-GoPackager/releases/tag/v1.0.0>

Download dungeondraft-pack.exe to your “DDScripts” folder.

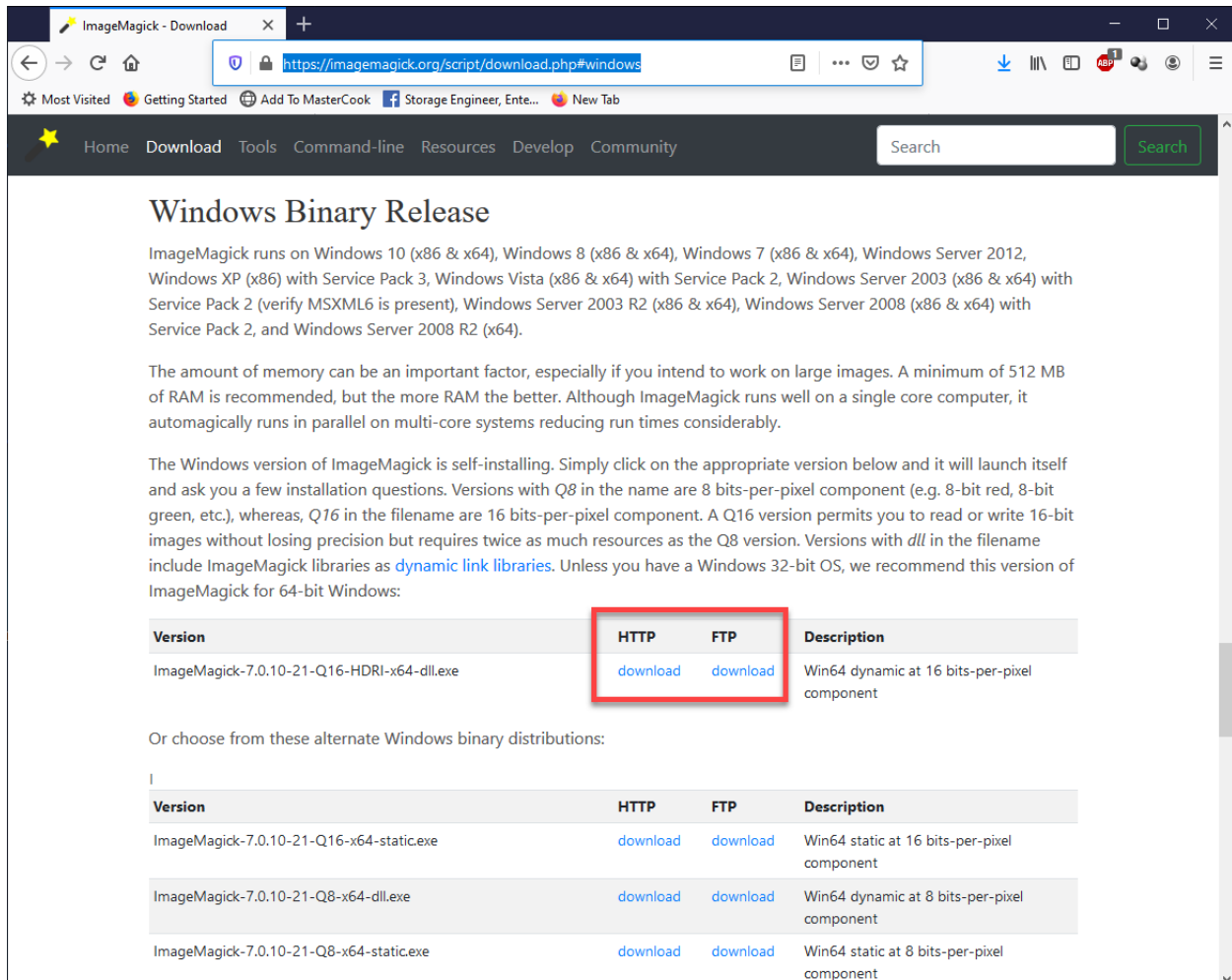
Download dungeondraft-unpack.exe to your “DDScripts” folder.

You just need those two files from that site. You don’t need any of the “Source code” files. You can ignore those.

Install ImageMagick

Browse to <https://imagemagick.org/script/download.php#windows>

Click on one of the download links for the top version. It should have a description of “Win64 dynamic at 16 bits-per-pixel component”. After downloading, run the downloaded file to install ImageMagick.



Windows Binary Release

ImageMagick runs on Windows 10 (x86 & x64), Windows 8 (x86 & x64), Windows 7 (x86 & x64), Windows Server 2012, Windows XP (x86) with Service Pack 3, Windows Vista (x86 & x64) with Service Pack 2, Windows Server 2003 (x86 & x64) with Service Pack 2 (verify MSXML6 is present), Windows Server 2003 R2 (x86 & x64), Windows Server 2008 (x86 & x64) with Service Pack 2, and Windows Server 2008 R2 (x64).

The amount of memory can be an important factor, especially if you intend to work on large images. A minimum of 512 MB of RAM is recommended, but the more RAM the better. Although ImageMagick runs well on a single core computer, it automatically runs in parallel on multi-core systems reducing run times considerably.

The Windows version of ImageMagick is self-installing. Simply click on the appropriate version below and it will launch itself and ask you a few installation questions. Versions with *Q8* in the name are 8 bits-per-pixel component (e.g. 8-bit red, 8-bit green, etc.), whereas, *Q16* in the filename are 16 bits-per-pixel component. A *Q16* version permits you to read or write 16-bit images without losing precision but requires twice as much resources as the *Q8* version. Versions with *dll* in the filename include ImageMagick libraries as [dynamic link libraries](#). Unless you have a Windows 32-bit OS, we recommend this version of ImageMagick for 64-bit Windows:

Version	HTTP	FTP	Description
ImageMagick-7.0.10-21-Q16-HDRI-x64-dll.exe	download	download	Win64 dynamic at 16 bits-per-pixel component

Or choose from these alternate Windows binary distributions:

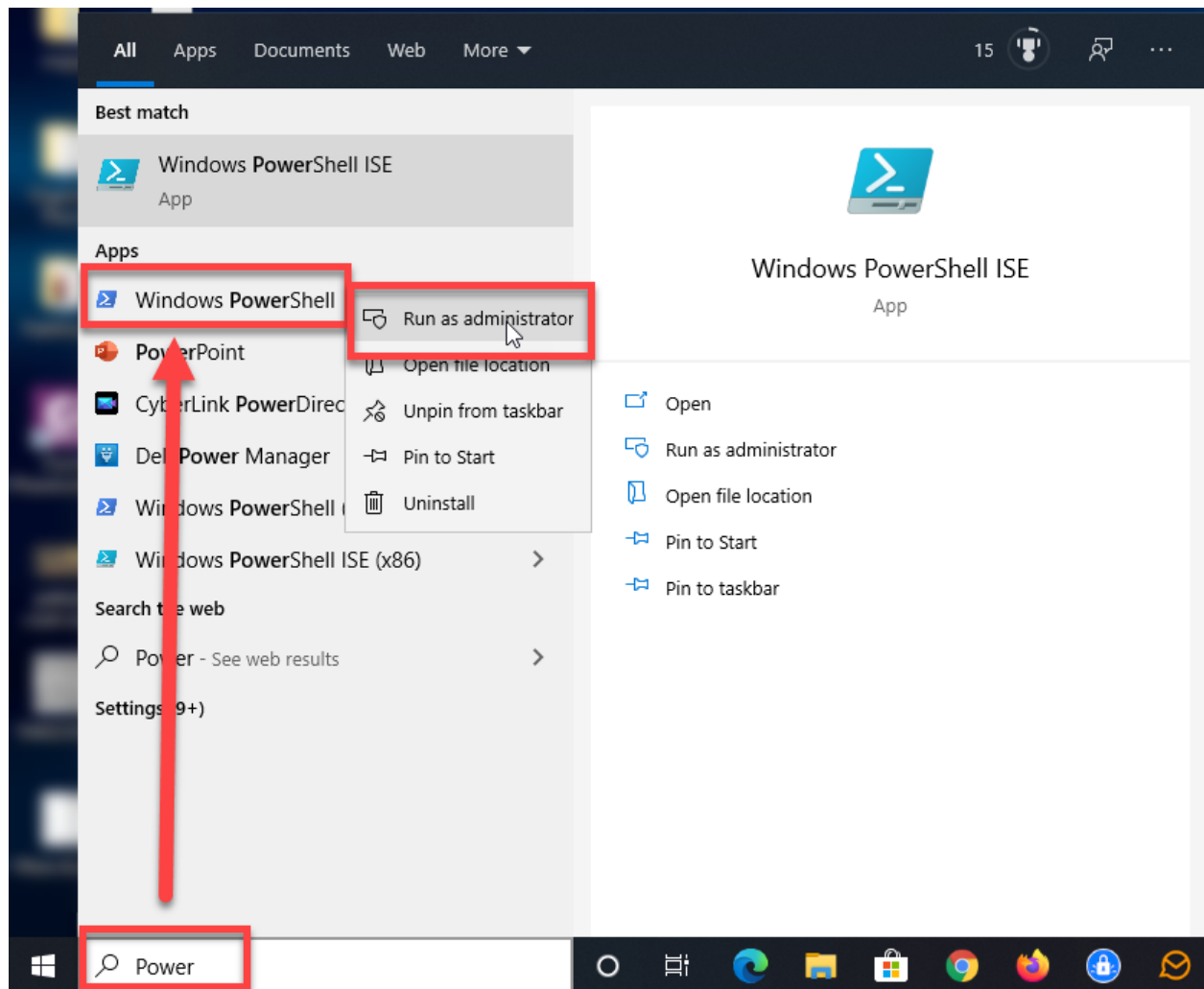
Version	HTTP	FTP	Description
ImageMagick-7.0.10-21-Q16-x64-static.exe	download	download	Win64 static at 16 bits-per-pixel component
ImageMagick-7.0.10-21-Q8-x64-dll.exe	download	download	Win64 dynamic at 8 bits-per-pixel component
ImageMagick-7.0.10-21-Q8-x64-static.exe	download	download	Win64 static at 8 bits-per-pixel component

How Do I Use These Scripts?

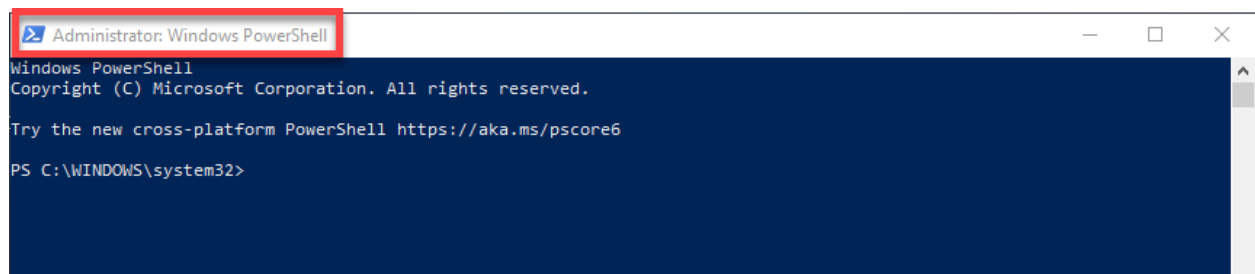
Set the PowerShell Execution Policy

Open Windows PowerShell by typing the name in the Windows search box (not Windows PowerShell ISE, just “Windows PowerShell”)

When “Windows PowerShell” appears, right-click it, and select “Run as administrator”.



When the PowerShell window opens, verify that the title bar says “Administrator: Windows PowerShell”.



Type the following command:

```
Get-ExecutionPolicy
```

This will tell you what your current execution policy is for PowerShell scripts. Unless you’ve changed this at some point, chances are that your execution policy is either “Default” or “Restricted”.

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> Get-ExecutionPolicy
Restricted
PS C:\WINDOWS\system32>
```

You'll need to change this to RemoteSigned. You can do this by typing the following command.

```
Set-ExecutionPolicy RemoteSigned
```

You will be asked to confirm the change.

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> Get-ExecutionPolicy
Restricted
PS C:\WINDOWS\system32> Set-ExecutionPolicy RemoteSigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
https://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"):
```

Press Y for Yes, then press ENTER.

Type `Get-ExecutionPolicy` again to verify that the policy changed.

```
Administrator: Windows PowerShell
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> Get-ExecutionPolicy
Restricted
PS C:\WINDOWS\system32> Set-ExecutionPolicy RemoteSigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
https://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): y
PS C:\WINDOWS\system32> Get-ExecutionPolicy
RemoteSigned
PS C:\WINDOWS\system32>
```

General PowerShell Syntax

Each script can be run by typing the full path and name of the script. For the sake of simplicity in this document, I will only refer to the script name (such as "DDTagAssets.ps1"), but in order to run the script, you need to refer to it by its full path. For example "C:\Users\EightBitz\DDScripts\DDTagAssets.ps1"

This is then followed by a series of parameters and values. For example, DDCopyAssets.ps1 can be run with a -Source parameter, and a -Destination parameter.

```
DDCopyAssets.ps1 -Source "C:\Users\EightBitz\CC3 Assets\Castles" -  
Destination "C:\Users\EightBitz\DD Assets\Castles"
```

You would, of course, type this command without the line break. I only included the line break here to make it readable. This will copy assets from the specified "CC3 Assets" folder to the specified "DDAssets" folder.

Each script has its own set of parameters, some of which are required, and some of which are optional. These will be explained for each script.

Windows PowerShell vs. Windows PowerShell ISE

I generally find it easier to use the Windows PowerShell ISE over PowerShell.

You can type your command in the top half and click the green play button on the top bar to run the command.

You can see the output and any error messages in the bottom half. If anything goes wrong, if you've made any kind of a typo or anything, you still have the command in the top half where you can make a simple edit and just click the green play button again.

Using the regular PowerShell window, you can press the up arrow on your keyboard to retrieve the last command you typed and edit it in place. That's just a little bit more awkward, though.

DDTagAssets.ps1

Description

This script will create a "default.dungeondraft_tags" file for folders that you intend to package with Dungeondraft.

It assumes that each folder you intend to package, each custom pack folder, is stored in a central parent folder.

It assumes that each custom pack folder contains a subfolder for each tag.

It assumes that colorable assets are each stored in a "colorable" folder within its corresponding subfolder.

For instance, a folder structure could look like this.

```
Custom Pack Folders  
  City Pack  
    textures  
      objects  
        Colorable  
        Fountains  
          Colorable  
          Houses  
            Colorable  
            Shops  
              Colorable  
  Dungeon Pack  
    textures
```

```

        objects
            Colorable
            Bones
                Colorable
            Rubble
                Colorable
            Traps
                Colorable
Nature Pack
    textures
        objects
            Colorable
            Bushes
                Colorable
            Flowers
                Colorable
            Trees
                Colorable

```

For the City Pack, anything under the “Fountains” subfolder will be tagged as “Fountains”. Anything under the “Houses” subfolder will be tagged as “Houses”, and anything under the “Shops” subfolder will be tagged as “Shops”.

Anything under any of the “Colorable” folders will also be tagged as “Colorable”. In order for these objects to actually be colorable in Dungeondraft, they still have to be (or contain parts that are) an appropriate shade red (anything where blue and green are within 10 RGB values, while red is 10+% higher than blue or green).

You can have files within subfolders within subfolders, but all files, regardless of how deep they are in the subfolder structure, will be tagged only according to the first level of subfolders under textures\objects. For instance, if you have different subfolders under “Nature Pack\textures\objects\Trees” named “Small” and “Large”, any files in these subfolders will still be tagged only as “Trees”.

Parameters

-Source <source path>

This is a required parameter. It defines the source folder that contains the custom pack folders for which you want to create default.dungeondraft_tags files.

```
DDTagAssets.ps1 -Source "Custom Pack Folders"
```

This example will tag all folders within “Custom Pack Folders”

-Include <comma-separated list of folder names>

This is an optional parameter.

If this parameter is omitted, all folders within the Source folder will be tagged.

If this parameter is given a value of *, all folders within the Source folder will be tagged.

Otherwise, this parameter should be a comma-separated list of which folders within the Source folder are to be tagged.

```
DDTagAssets.ps1 -Source "Custom Pack Folders" -Include *
```

This example is equivalent to omitting the -Include parameter. It will tag all folders within "Custom Pack Folders".

```
DDTagAssets.ps1 -Source "Custom Pack Folders" -Include "City Pack"
```

This example will look for the "City Pack" folder in "Custom Pack Folders" and tag only that folder, ignoring the others.

```
DDTagAssets.ps1 -Source "Custom Pack Folders" -Include "City  
Pack,Dungeon Pack"
```

This example will look for the "City Pack" and "Dungeon Pack" folders in "Custom Pack Folders" and tag only those two folders, ignoring the others.

-Exclude <comma-separated list of folder names>

This is an optional parameter. This parameter will only take effect if all folders are included. Otherwise, this parameter will be ignored.

This specifies a list of folders to exclude from tagging.

```
DDTagAssets.ps1 -Source "Custom Pack Folders" -Exclude "City Pack"
```

This example will tag all folders in "Custom Pack Folders" except for the "City Pack" folder.

```
DDTagAssets.ps1 -Source "Custom Pack Folders" -Exclude "City  
Pack,Dungeon Pack"
```

This example will tag all folders in "Custom Pack Folders" except for the "City Pack" and "Dungeon Pack" folders.

-DefaultTag <string>

This is an optional parameter. This parameter defines a default tag for objects that are in the "textures\objects" folder of a given pack folder (i.e. not in any subfolders). Otherwise, these objects will not be tagged.

```
DDTagAssets.ps1 -Source "Custom Pack Folders" -Include "City Pack" -  
DefaultTag "Miscellaneous"
```

This example will tag the "City Pack" folder in "Custom Pack Folders", and any objects in the root of the "textures\objects" folder will be tagged as "Miscellaneous".

[DDCopyAssets.ps1](#)

Description

This script takes assets from a specified source, and it copies them into a specified destination with a folder structure that is ready-to-be-packed by Dungeondraft.

In particular, for CC3+ assets, it only copies one version of each asset (the highest quality version), and it optionally copies assets whose names start with “Door ” or “Window ” to “textures\portals” instead of “textures\objects”.

Parameters

-Source <source path> **-Destination** <destination path>

These are both required parameters.

```
DDCopyAssets.ps1 -Source "CC3\Castles" -Destination "DD\Castles"
```

This example will copy supported image files from “CC3\Castles” to “DD\Castles\textures\objects” while also replicating the original folder structure from “CC3\Castles” to “DD\Castles\textures\objects”.

As the -Portals parameter is true by default, this example will also copy doors and windows to “DD\Castles\textures\portals” instead of to “DD\Castles\textures\portals”.

-CreateTagFile True or False

This is an optional parameter. Its default value is False.

This parameter tells the script whether or not to create a default.dungeondraft_tags file in the destination folder by calling the DDTAGAssets.ps1 script.

```
DDCopyAssets.ps1 -Source "CC3\Castles" -Destination "DD\Castles" -  
CreateTagFile True
```

This example will create the default.dungeondraft_tags file after copying the files.

```
DDCopyAssets.ps1 -Source "CC3\Castles" -Destination "DD\Castles" -  
CreateTagFile False
```

This example will not create the default.dungeondraft_tags file. This is the default behavior. This is equivalent to omitting the -CreateTagFile parameter.

-Portals True or False

This is an optional parameter. Its default value is True.

This parameter tells the script whether or not to copy doors and windows to “textures\portals” instead of “textures\objects”.

```
DDCopyAssets.ps1 -Source "CC3\Castles" -Destination "DD\Castles" -  
Portals True
```

This example will copy doors and windows to “textures\portals”. This is the default behavior. This is equivalent to omitting the -Portals parameter.

```
DDCopyAssets.ps1 -Source "CC3\Castles" -Destination "DD\Castles" -  
Portals False
```

This example will copy doors and windows to “textures\objects”.

```
DDCopyAssets.ps1 -Source "CC3\Castles" -Destination "DD\Castles" -  
Portals False -CreateTagFile True
```


This example will copy doors and windows to “textures\objects”, then call the DDTagAssets.ps1 script to create a default.dungeondraft_tags file.

DDConvertAssets.ps1

Description

This script converts images in the specified source folder to webp images in the specified destination folder. Any source images that are already in webp format will simply be copied to the destination.

In addition, any Dungeondraft data files will be modified so that all file entries have their extensions changed.

This script uses ImageMagick to convert images to webp.

<https://imagemagick.org/script/download.php#windows>

Parameters

-Source <source path>

This is a required parameter. It defines the source folder that contains the images to be converted.

If the -Source parameter is used on its own, with no -Destination parameter, the destination will default to the name of the source folder appended with “ - webp”. So if the value for -Source is “My Asset Folders”, the default value for the destination will be “My Asset Folders - webp”.

```
DDConvertAssets.ps1 -Source "My Asset Folders"
```

This example will copy the “My Asset Folders” folder structure to “My Asset Folders - webp” while converting all image files to webp format and modifying .dungeondraft files to reflect the file name changes.

-Destination <source path>

This an optional parameter. It defines the destination folder that will contain the converted images.

```
DDConvertAssets.ps1 -Source "My Asset Folders" -Destination "My  
Converted Asset Folders"
```

This example will copy the “My Asset Folders” folder structure to “My Converted Assets” while converting all image files to webp format and modifying .dungeondraft files to reflect the file name changes.

```
DDConvertAssets.ps1 -Source "My Asset Folders" -Destination "Converted  
Assets\My Asset Folders"
```

This example will copy the “My Asset Folders” folder structure to “Converted Assets\My Asset Folders” while converting all image files to webp format and modifying .dungeondraft files to reflect the file name changes.

DDConvertPacks.ps1

Description

This script:

- runs dungeondraft-upack.exe to unpack .dungeondraft_pack files to a folder structure
- runs the DDConvertAssets.ps1 script to convert the images to webp format
- repacks the converted folders into new .dungeondraft_pack files.

This script creates three working folders within the destination folder.

- Unpacked Assets: this folder will contain the original folder structures of the packs with all images in their original formats.
- Converted Folders: this folder will contain the original folder structure of the packs with all images in webp format.
- Converted Packs: this folder will contain the repackaged packs created from the Converted Folders.

Parameters

-Source <source path>

This is a required parameter. It defines the source folder that contains the .dungeondraft_pack files to be converted, or the full path and filename of a single pack to be converted.

If the -Source parameter is used on its own, with no -Destination parameter, the destination will default to the name of the source folder appended with " - webp". So if the value for -Source is "My Asset Packs", the default value for the destination will be "My Asset Packs - webp".

```
DDConvertPacks.ps1 -Source "My Asset Packs"
```

This example will convert all the .dungeondraft_pack files in the "My Asset Packs" folder, storing the converted packs in "My Asset Packs - webp\Converted Packs". The other two working folders "Unpacked Assets" and "Converted Folders" will be removed, leaving only "Converted Packs".

```
DDConvertPacks.ps1 -Source "My Asset Packs\City  
Pack.dungeondraft_pack"
```

This example will convert the "City Pack.dungeondraft_pack" file in the "My Asset Packs" folder, storing the converted pack in "My Asset Packs - webp\Converted Packs". The other two working folders "Unpacked Assets" and "Converted Folders" will be removed, leaving only "Converted Packs".

-Destination <destination path>

This an optional parameter. It defines the destination folder that will contain the converted packs.

```
DDConvertPacks.ps1 -Source "My Asset Packs" -Destination "My Converted  
Packs"
```

This example will convert all the .dungeondraft_pack files in the "My Asset Packs" folder, storing the converted packs in "My Converted Packs\Converted Packs". The other two working folders "Unpacked Assets" and "Converted Folders" will be removed, leaving only "Converted Packs".

```
DDConvertPacks.ps1 -Source "My Asset Packs" -Destination "Converted  
Packs\My Asset Packs"
```

This example will convert all the .dungeondraft_pack files in the “My Asset Packs” folder, storing the converted packs in “Converted Packs\My Asset Packs\Converted Packs”. The other two working folders “Unpacked Assets” and “Converted Folders” will be removed, leaving only “Converted Packs”.

-Include <comma-separated list of package names>

This is an optional parameter.

If this parameter is omitted, all packs within the Source folder will be converted.

If this parameter is given a value of *, all packs within the Source folder will be converted.

Otherwise, this parameter should be a comma-separated list of which packs within the Source folder are to be converted.

```
DDConvertPacks.ps1 -Source "My Asset Packs" -Include *
```

This example is equivalent to omitting the -Include parameter. It will convert all packs within “My Asset Packs”.

```
DDConvertPacks.ps1 -Source "My Asset Packs" -Include "City  
Pack.dungeondraft_pack"
```

This example will look for the “City Pack.dungeondraft_pack” file in “My Asset Packs” and convert only that pack file, ignoring the others.

```
DDConvertPacks.ps1 -Source "My Asset Packs" -Include "City  
Pack.dungeondraft_pack,Dungeon Pack.dungeondraft_pack"
```

This example will look for the “City Pack.dungeondraft_pack” and “Dungeon Pack.dungeondraft” packs in “My Asset Packs” and convert only those two pack files, ignoring the others.

-Exclude <comma-separated list of package names>

This is an optional parameter. This parameter will only take effect if all pack files are included.

Otherwise, this parameter will be ignored.

This specifies a list of pack files to exclude from conversion.

```
DDConvertPacks.ps1 -Source "My Asset Packs" -Exclude "City  
Pack.dungeondraft_pack"
```

This example will convert all packs in the “My Asset Packs” folder except for the “City Pack.dungeondraft_pack” file.

```
DDConvertPacks.ps1 -Source "My Asset Packs" -Exclude "City  
Pack.dungeondraft_pack,Dungeon Pack.dungeondraft_pack"
```

This example will convert all packs in “My Asset Packs” except for the “City Pack” and “Dungeon Pack” packs.

-CleanUp True or False

This is an optional parameter.

```
DDConvertPacks.ps1 -Source "My Asset Packs" -CleanUp False
```

This example will convert all the .dungeondraft_pack files in the “My Asset Packs” folder, storing the converted packs in “My Asset Packs - webp\Converted Packs”. All working folders will be left in place so that “My Asset Packs - webp” will also contain the “Unpacked Assets” and “Converted Folders” folders.