Appendix B: The Content Pipeline

Every MonoGame game we write will have *content*. Content includes the graphics you include in your game, sound effects, text fonts, and so on. Although you can include many content assets in their raw form (.png files for graphics, .wav files for sounds, etc.) in your monoGame projects, it's better to convert those assets into .xnb format instead.

Why should we do that? There's a great, detailed discussion about that here (see MonoGame - Content Pipeline ... Why???) , but the big idea is that different target platforms (Windows, Android, iOS, etc.) have different default compression formats and handle the standard raw formats differently, so it can be more memory efficient and/or CPU efficient to convert those assets to .xnb before deploying your game to the target platform. This is called the content pipeline because you can think of this as moving your content through a pipe of processing steps from the artist/musician/level designer (and the raw asset files they generate) to your MonoGame solution.

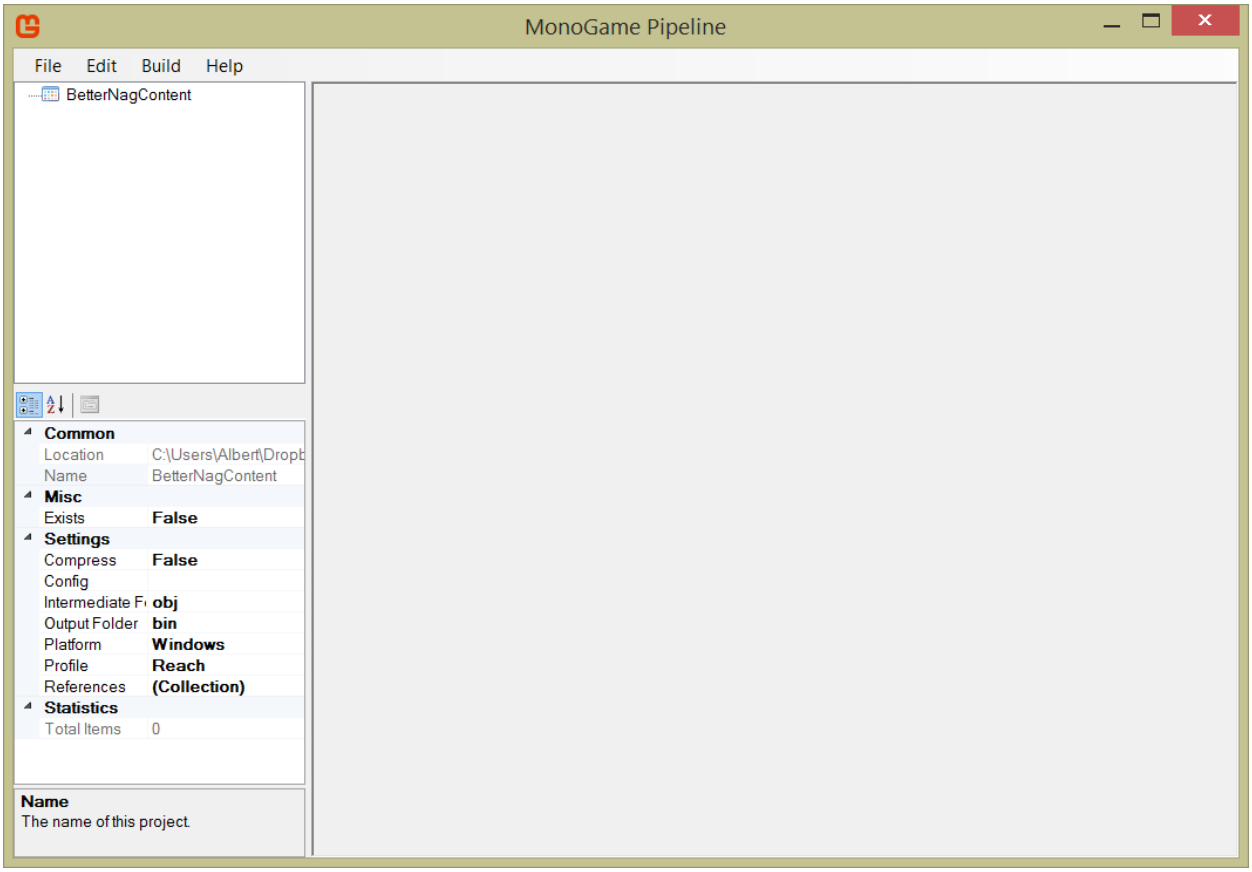
The good news is that MonoGame includes a Pipeline tool you can use to do these conversions. The tool is located in the C:\Program Files (x86)\MSBuild\MonoGame\v3.0\Tools\ folder on my computer; don't forget the MSBuild part of the path.

Because you want your .xnb files to be optimized for your target platform, you'll want to build your content separately for each of the target platforms you plan to deploy to. That's a small price to pay to optimize the run-time performance of your game, though! We'll use Windows as our target platform throughout this book.

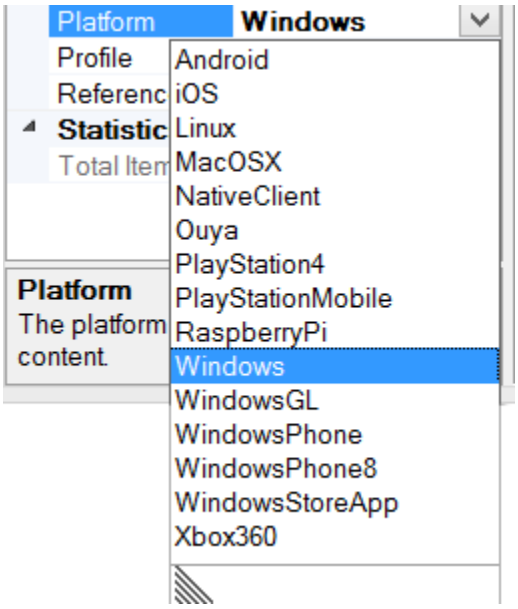
B.1. Converting Graphical Assets

The first place we use graphical assets in this book is in Chapter 5 in Section 5.3. We'll go through the steps we follow for that example here.

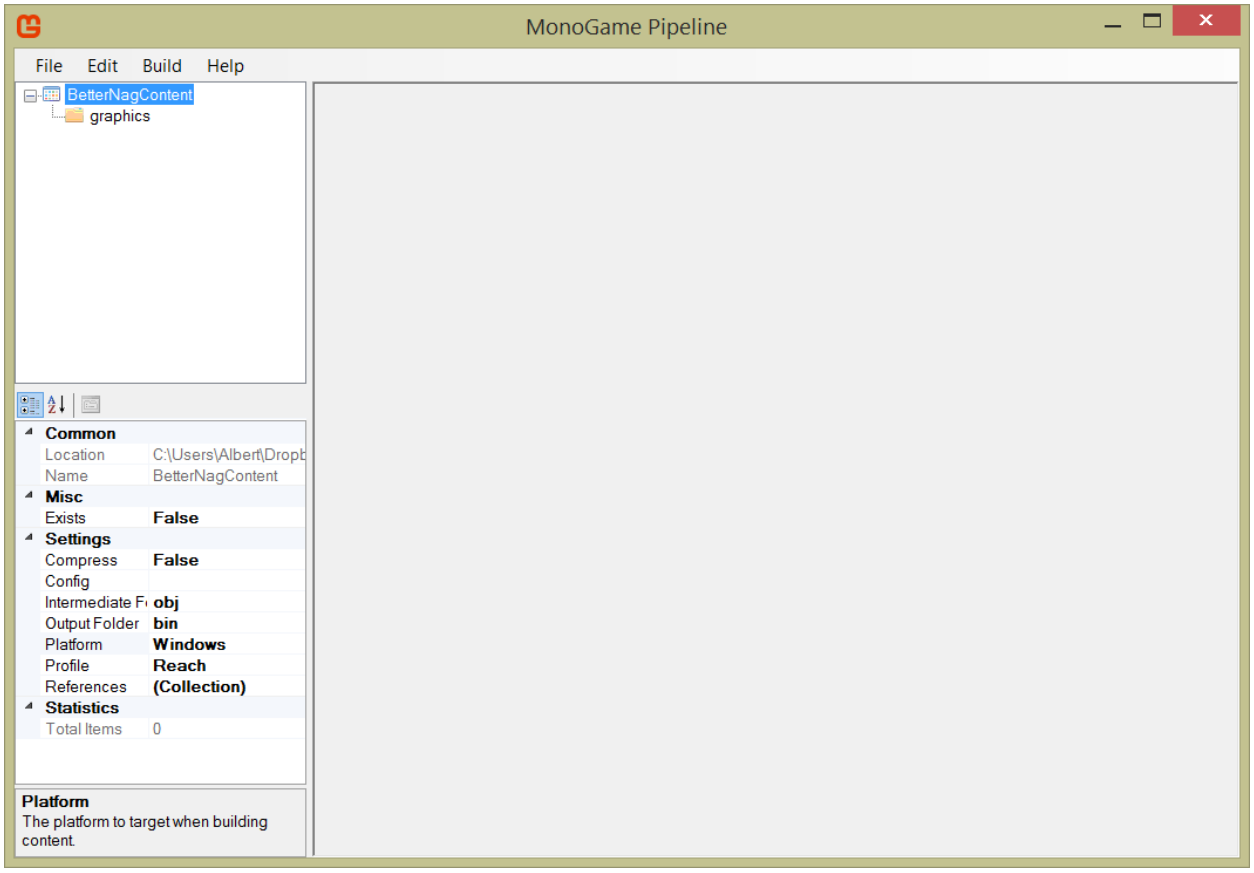
When we start the Pipeline tool, we select the name and location of our project, then click the Save button. As you can see below, I called my project BetterNagContent.



If you left click the top of the project tree (BetterNagContent in the upper left pane), the lower left pane gives you a number of configuration options for the project. Clicking Windows for the Platform and then clicking the down arrow on the right gives the set of target platforms shown below (you might have to drag the lower left corner of the popup to see the full list). How awesome is that!

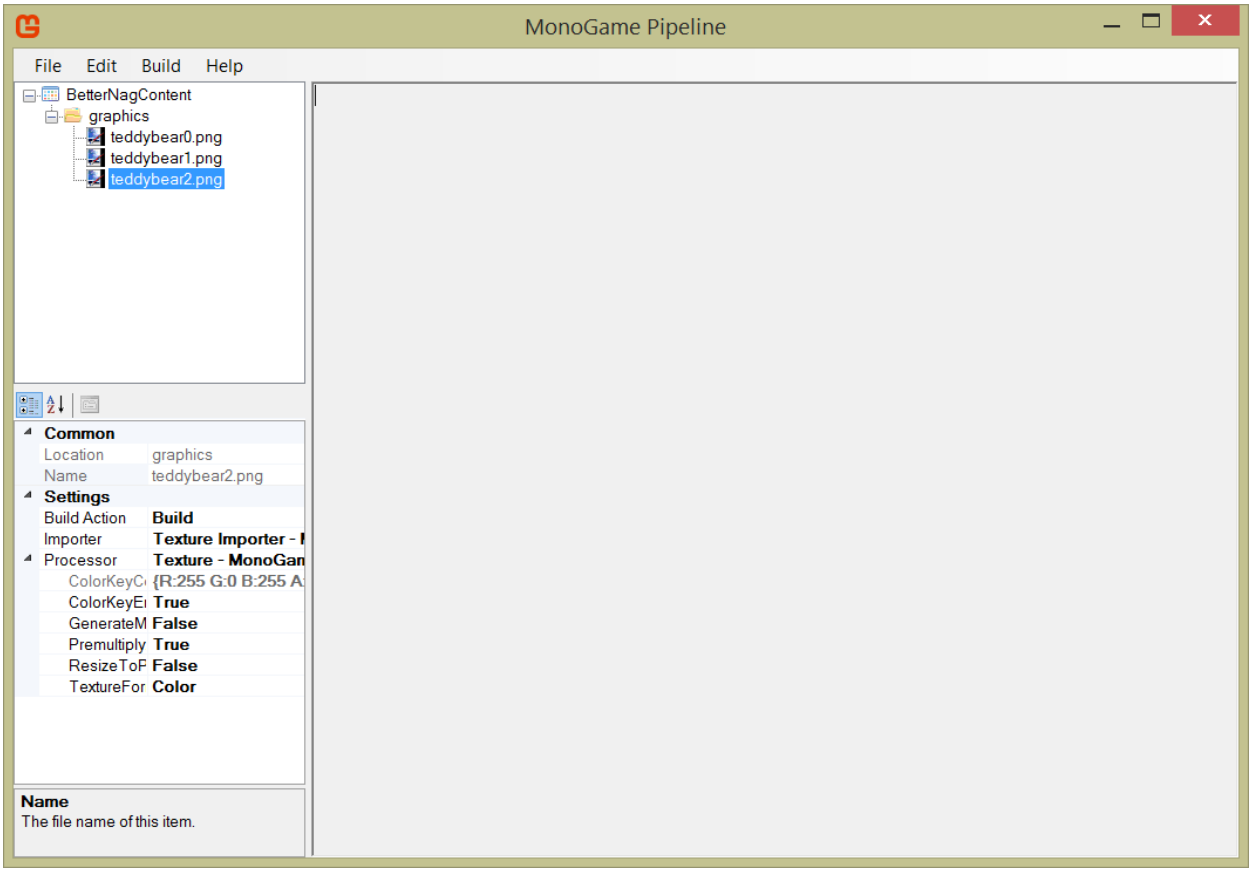


Let's start adding content to our pipeline project. Although we could just include all our content at the top level, we've found that it's better for larger games to include folders to hold each type of asset. Although that will feel like a bit of overkill in your early games, we might as well just do it the same way all the time. Right click BetterNagContent in the upper left pane and select Add > New Folder ..., name the folder graphics, and click the OK button.

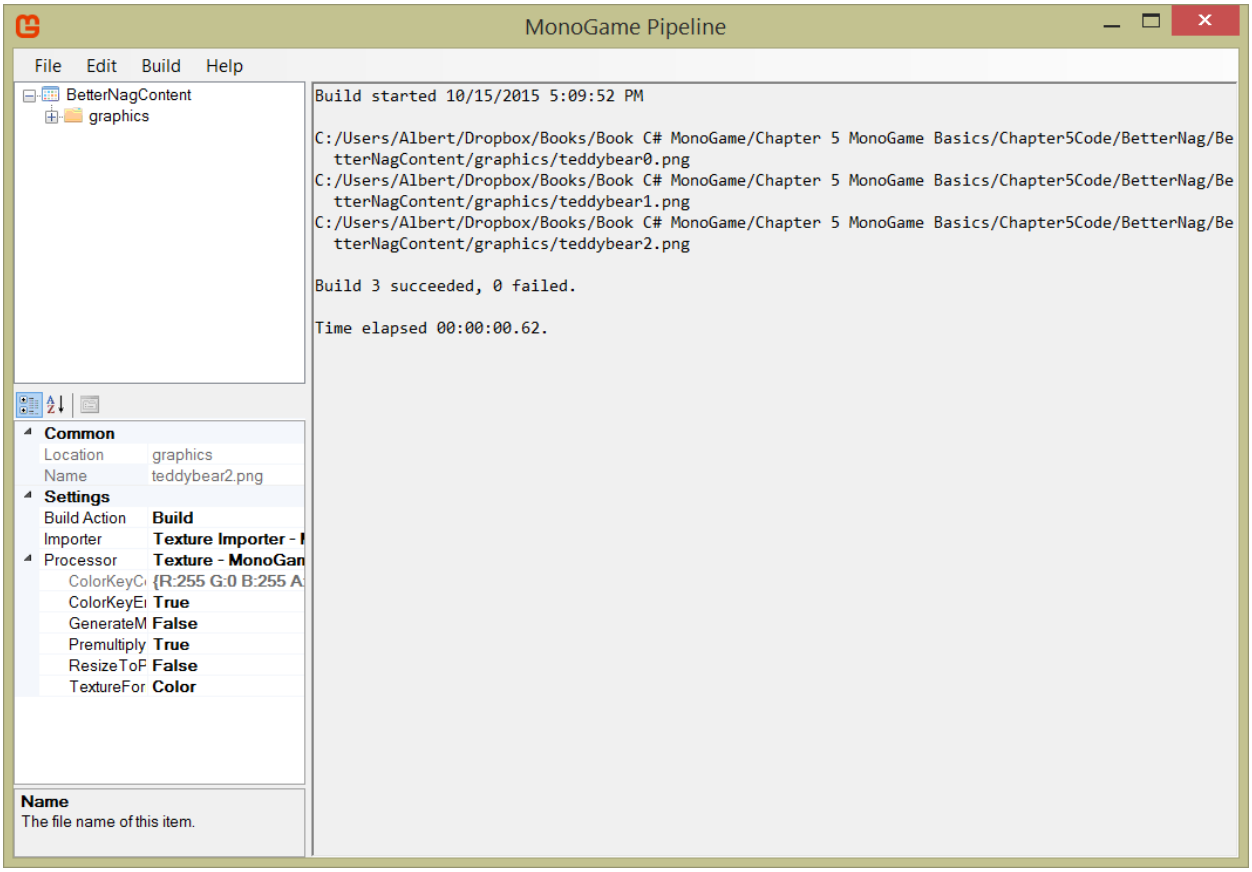


Now we need to copy our .png files into the folder we just created. Go to Windows Explorer and copy those files over into the new graphics folder. Although MonoGame supports a variety of graphics formats, we just always use .png files in our games.

At this point, the .png files are actually in the correct folder, but they're not included in our pipeline project yet. Right click the graphics folder in the upper left pane and select Add > Existing Item ... Select all three of the .png files in the popup and click the Open button. If you click the + to the left of the graphics folder to expand the folder, you'll see those 3 files have now been added to the project.



We now have all the files added to our project, so it's time to convert them to .xnb format. Press F6 or select Build > Build from the top menu bar to do the conversion; you should get a message in the right pane similar to the one shown below. The important part of the message is near the end, where we can see that 3 succeeded and 0 failed.

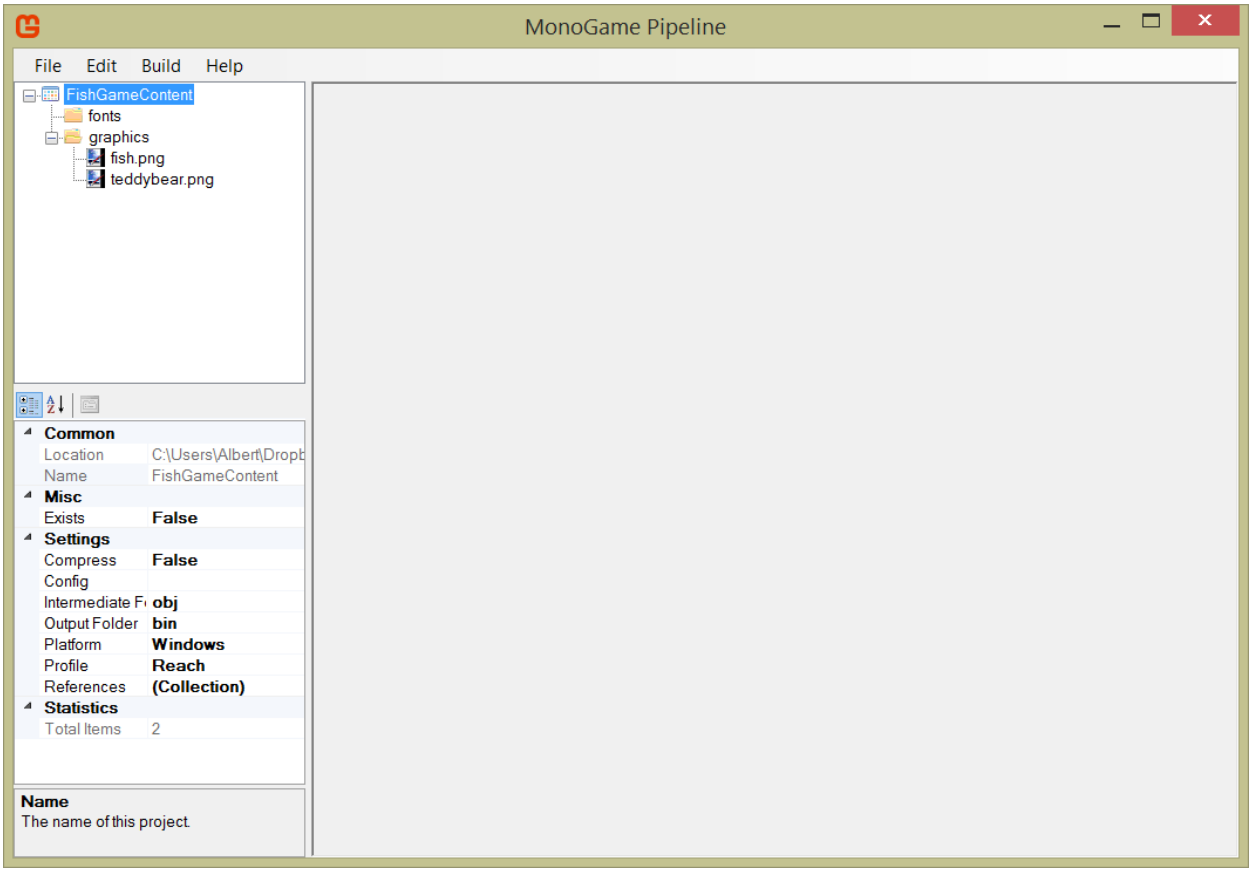


The last thing you need to know if where the .xnb files ended up. For this example, they're in the <pipeline project location>\bin\graphics folder. You need to know where those files are so you can include them in your MonoGame solution; we cover how to do that in Section 5.3.

B.2. Converting Sprite Fonts

Linux users: As of this writing, the Pipeline tool fails when trying to build xnb files from spritefont files. The tool crashes because it can't find the dll for the Nvidia Texture Tools (which is strange because dlls are for Windows, not Linux). You'll have to use the process below to build the spritefont content on Windows or OS X and then copy it over.

The first place we use sprite fonts in this book is in Chapter 14 in Section 14.2, where we add a score display to a small game. We start with the pipeline project we created for that game; the first thing we do is add a fonts folder to the project:



Adding a sprite font to our project is different from what we've done with the Pipeline tool so far. Up to this point, we've simply been adding existing items (.png and .wav files) to our projects. This time, we'll be adding a new item instead.

Right click the fonts folder and select Add > New Item ... Left click Sprite Font Description, name the sprite font Arial, and click the OK button. Because you probably want to edit the characteristics of the new sprite font (click the + to the left of the fonts folder to see it), double click the Arial.spritefont file. Windows doesn't know how to open a .spritefont file by default, so you'll need to pick the application you use to edit it (I use Notepad++, but any text editor will work fine).

Change the size of the font to 14, save the file, and close your text editor. Build the project to convert everything to .xnb format.

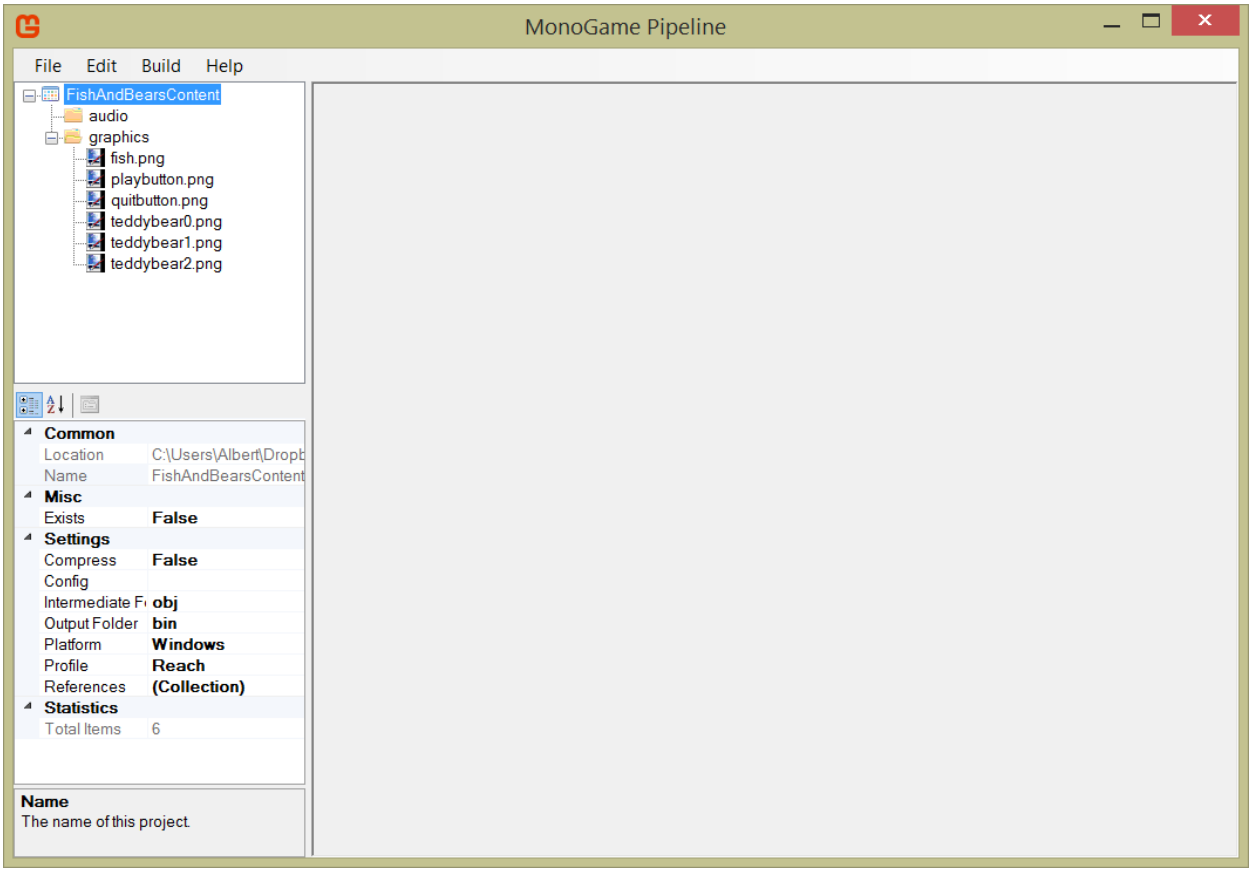
For this example, the converted spritefont .xnb file is in the <pipeline project location>\bin\fonts folder. You need to know where that file is so you can include it in your MonoGame solution; we cover how to do that in Chapter 14.

B.3. Converting Audio Assets

Mac and Linux users: As of this writing, the Pipeline tool fails when trying to build xnb files from wav files. Although you could use the process below to build the audio content on a

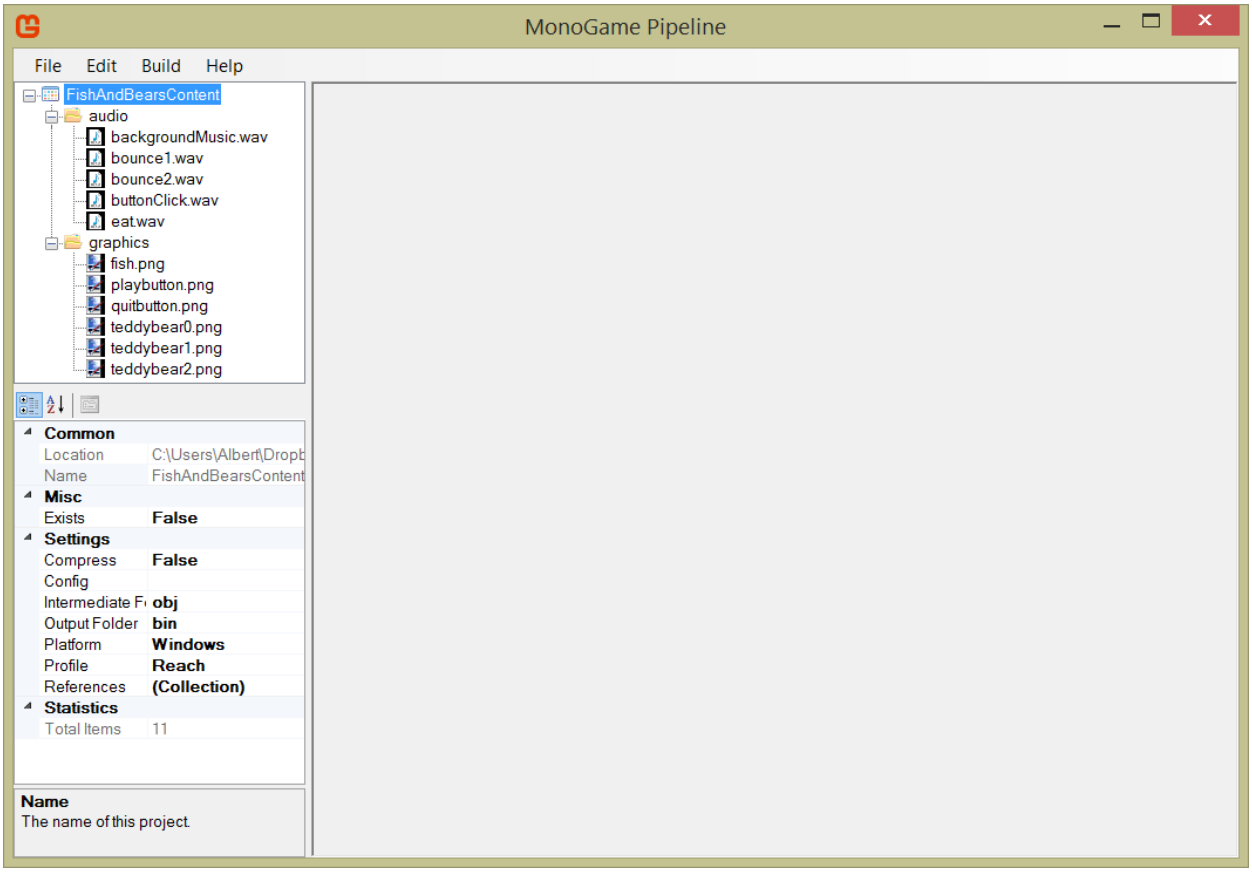
Windows machine and then copy it over, we discuss an easier way to include audio content in your game in Chapter 15. Windows users, read on!

The first place we use audio assets in this book is in Chapter 15, where we add audio to our Fish and Bears game. We start with the pipeline project we created for that game; the first thing we do is add an audio folder to the project:



Now we need to copy our .wav files into the folder we just created. Go to Windows Explorer and copy those files over into the new audio folder. Although MonoGame supports a variety of audio formats, we usually use .wav files in our games.

Just as with our .png files in the previous section, the .wav files are in the correct folder, but they're not included in our pipeline project yet. Right click the audio folder in the upper left pane and select Add > Existing Item ... Select all of the .wav files in the popup and click the Open button. If you click the + to the left of the audio folder to expand the folder, you'll see those files have now been added to the project.

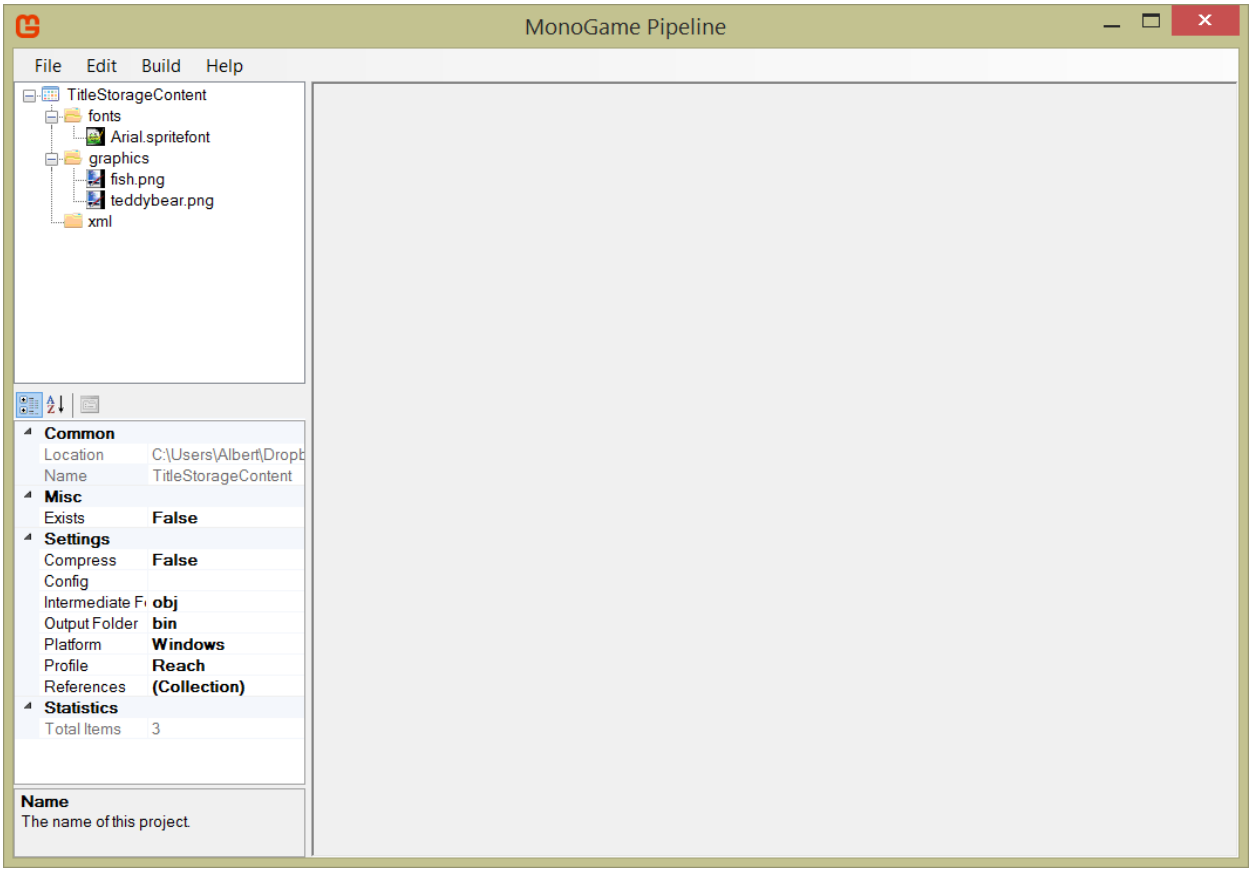


Press F6 or select Build > Build from the top menu bar to convert all the files to .xnb format. The message you get in the right pane should say 11 succeeded and 0 failed near the end.

For this example, the converted audio .xnb files are in the <pipeline project location>\bin\audio folder. You need to know where those files are so you can include them in your MonoGame solution; we cover how to do that in Chapter 15.

B.4. Converting XML Files

The first place we use XML files in this book is in Chapter 18 in Section 18.4, where we learn how to use title storage. We start with the pipeline project we created for the game in Section 14.2; the first thing we do is add an xml folder to the project:

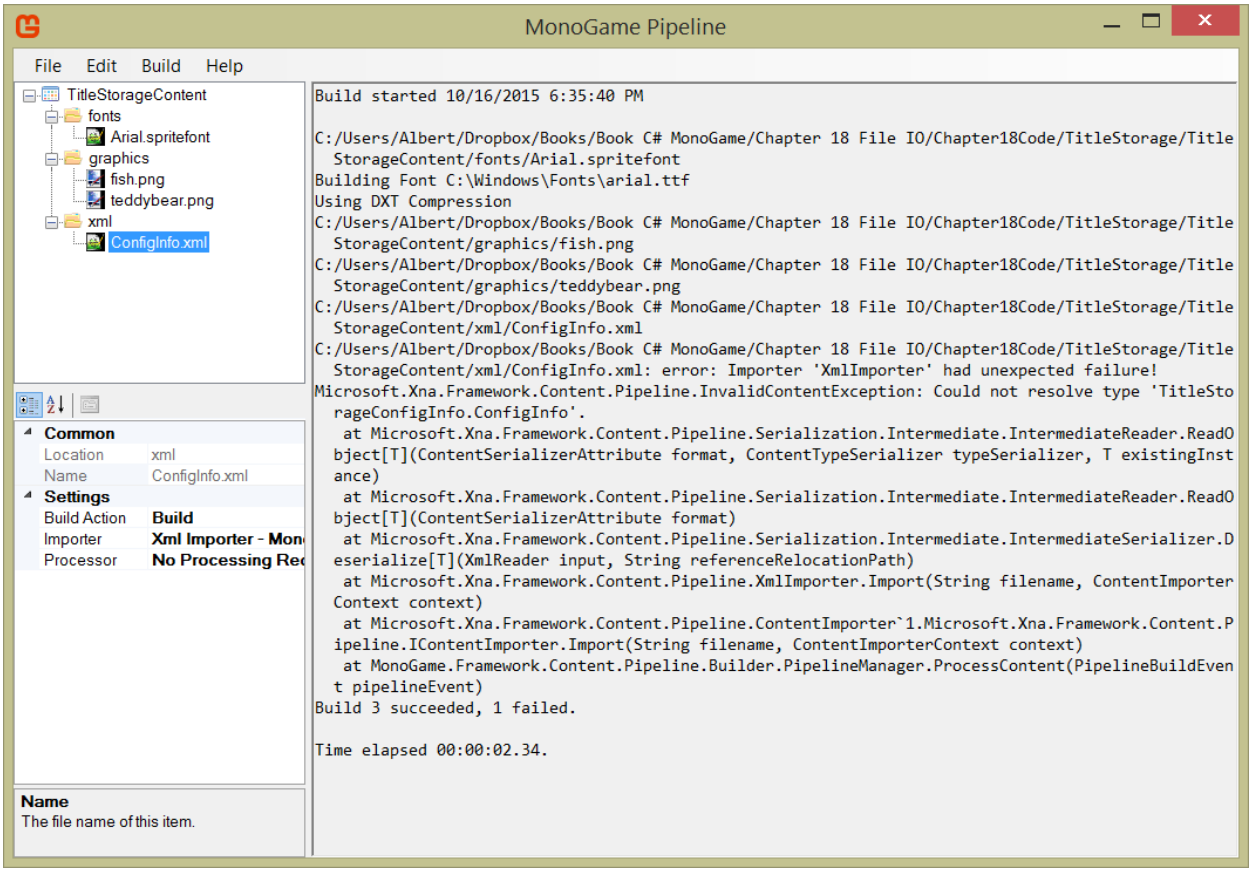


Right click the xml folder and select Add > New Item ... Left click XML Content, name the XML file ConfigInfo, and click the OK button. Because you probably want to edit the characteristics of the new XML file (click the + to the left of the xml folder to see it), double click the ConfigInfo.xml file. Your computer might open the file with a browser, but you need to be able to edit the file, so pick the application you want to use to edit it (I use Notepad++, but any text editor will work fine).

We'll discuss the different pieces of the XML file in Section 18.4, so for now edit the XML file to look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<XnaContent xmlns:ns="Microsoft.Xna.Framework">
  <Asset Type="TitleStorageConfigInfo.ConfigInfo">
      <FishMoveAmount>5</FishMoveAmount>
      <MinTeddySpeed>2</MinTeddySpeed>
      <MaxTeddySpeed>7</MaxTeddySpeed>
      <BearSpawnDelayMilliseconds>500</BearSpawnDelayMilliseconds>
      <BearScore>10</BearScore>
  </Asset>
</XnaContent>
```

Save the file. Unfortunately, when we press F6 to build our content, we get the following:

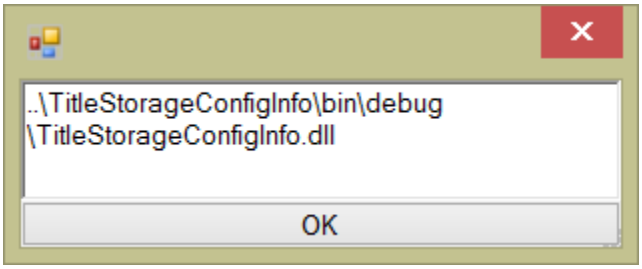


Darn, our first failed content build! If you look at the error message that starts at about the middle of the right pane, you can see that the Pipeline tool "Could not resolve type 'TitleStorageConfigInfo.ConfigInfo'." The problem is that in Section 18.4 we create a `ConfigInfo` class for our game configuration info and we included a reference to that class in our XML file, but the Pipeline tool doesn't know anything about that class.

The solution to our problem is to give the Pipeline tool a reference to the dll that contains the `ConfigInfo` class. Note that the following steps will only work after you've written and compiled the `ConfigInfo` class in Section 18.4.

Left click TitleStorageContent in the top left pane, click **(Collection)** to the right of References in the bottom left pane, then click the small box with the three dots in it that appears.

Here comes the tricky part – type in the path to the dll that contains the `ConfigInfo` class and click the OK button. Here's what the path looks like if you're using the code provided on the Burning Teddy web site:



Notice that we've used a relative path so nothing breaks if we decide to move our entire MonoGame solution to a different location; we suggest you use the same approach.

Now you should be able to build the pipeline project without any failures.