XNA Development: Tutorial 7

By Matthew Christian (Matt@InsideGamer.org)

Code and Other Tutorials Found at http://www.insidegamer.org/xnatutorials.aspx

In order to create a more realistic mood within a game, sound effects and music are used extensively. A creaking door or the sound of a cardboard box hitting the floor could strike the perfect mood and create the ultimate gaming experience. Of course, music behind menus and fight scenes always help immerse the player in the game. It's time for our XNA demo to include a class dedicated to holding audio items and managing them appropriately.

GameAudio Class Design

The GameAudio class is designed relatively similar to what we've built using the GameSprite class. We will use a list comprised of a custom audio structure we'll implement and have many different methods to interact with those audio items.

Many basic functions have been implemented in the GameAudio class (see Diagram 1). Of the most basic of these methods is the PlaySound(), PauseSound(), UnpauseSound(), and so on. Audio is quite simple when it comes to manipulating it, the trick is encapsulating it into a dynamic class.

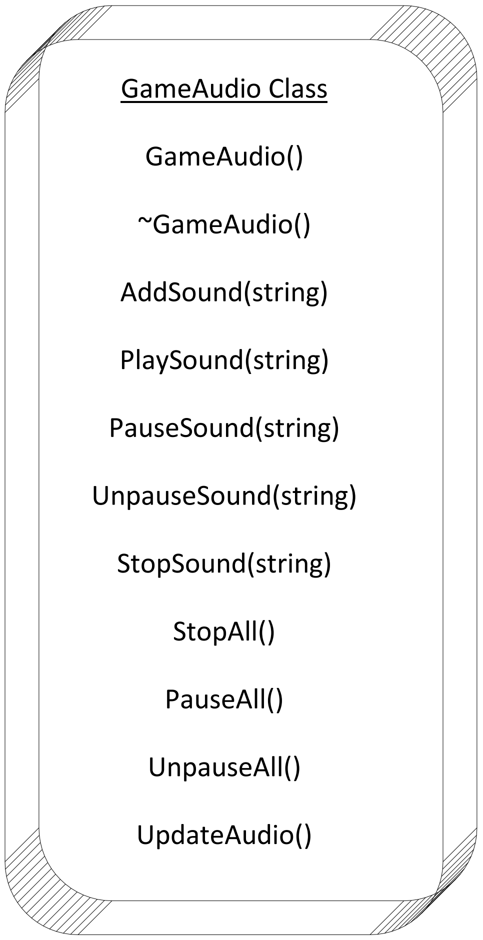


Diagram 1 –GameAudio Class

Using XACT

XNA uses a tool provided with the Game Studio installation to process audio within a game. This tool is the Microsoft Cross-Platform Audio Creation Tool (XACT). XACT allows you to add sound files, change information regarding their playback, and review the files themselves.

Why is XACT so special? Good question, but easy to answer. XACT, while allowing you all of the described above, lets you add your files, save the project, and will automatically build audio cues that you can reference in XNA. A cue can be thought of a as handle for a certain playback of a sound. I could have one file and many cues that all sound unique if I wanted thanks to XACT. Plus, when I need to access the cue in my code, all that's needed is a reference to that cue name and you've got sound.

Lets start first by using XACT. Go to your Start menu, click All Programs, then Microsoft XNA Game Studio 2.0, then Tools, then Microsoft Cross-Platform Audio Creation Tool (XACT). See Figure 1 for a better understanding.

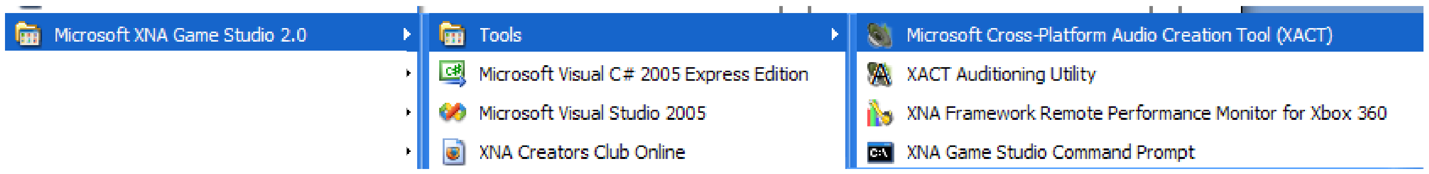

Microsoft XNA Game Studio 2.0
Tools
Microsoft Visual C# 2005 Express Edition
Microsoft Visual Studio 2005
XNA Creators Club Online
Microsoft Cross-Platform Audio Creation Tool (XACT)
XACT Auditioning Utility
XNA Framework Remote Performance Monitor for Xbox 360
XNA Game Studio Command Prompt

Figure 1 - Microsoft Cross-Platform Audio Creation Tool (XACT)

After it loads, you should see the default screen:

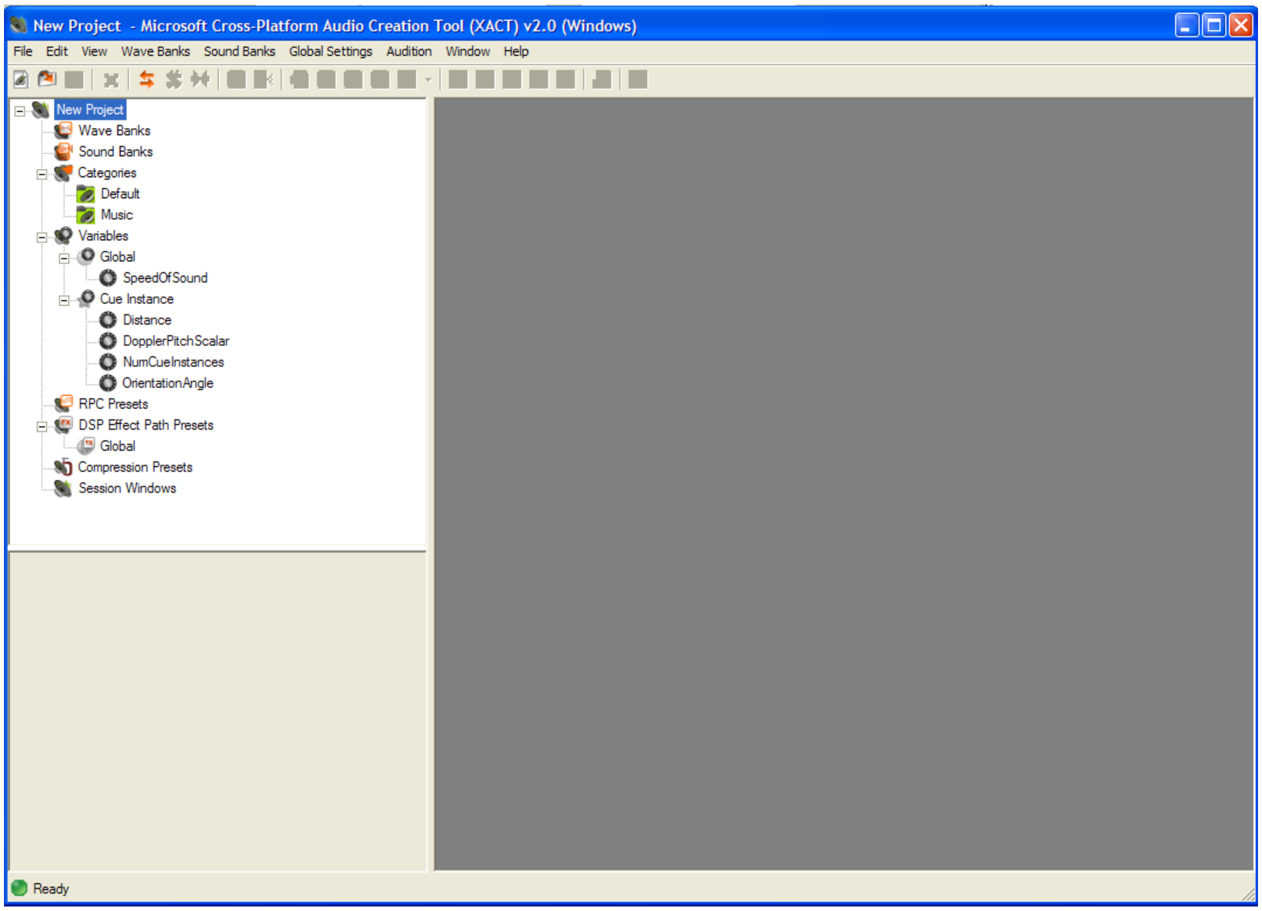


Figure 2 - Default XACT Window

We want to start a new project in XACT so click File > New Project. Now, it will ask you where you want to store the new project. Browse to the directory your solution file is in and create a new folder called ‘Audio Backup’. There’s a reason to this madness; if we started this project somewhere like ‘My Documents’ and added a sound file located on the desktop, the sound file would have a crazy extension attached which we don’t want. Eventually, all our sound files will be stored in the same directory as our XACT project. After creating the folder (Figure 3), name the project ‘GameSounds’ and click Save.

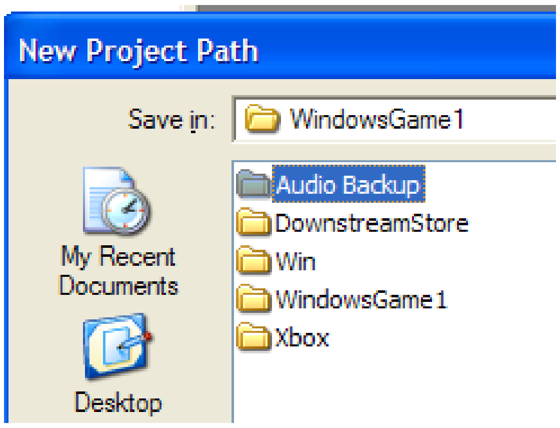


Figure 3 - Audio Backup Folder

Now, you should have a new XACT project named GameSounds and should be ready to create new items within it. Before we can add any sounds or cues, we need a Wave Bank and a Sound Bank. A Wave Bank is the place you will store your sounds in and, thanks to XACT, will be compiled down into one big file for use by XNA. While the Wave Bank is the collection of .wav files used, the Sound Bank is the collection of cues we can reference using XNA. On the left you should see two items near the top of the cascading menu titled Wave Bank and Sound Bank. Right click on Wave Bank and select 'New Wave Bank'. Do the same for the Sound Bank.

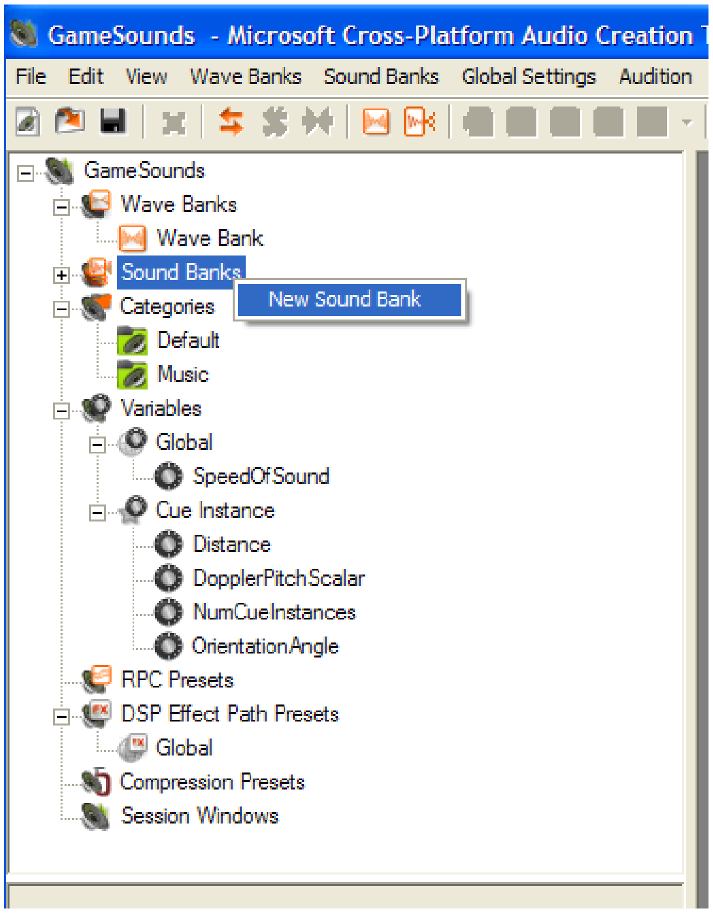
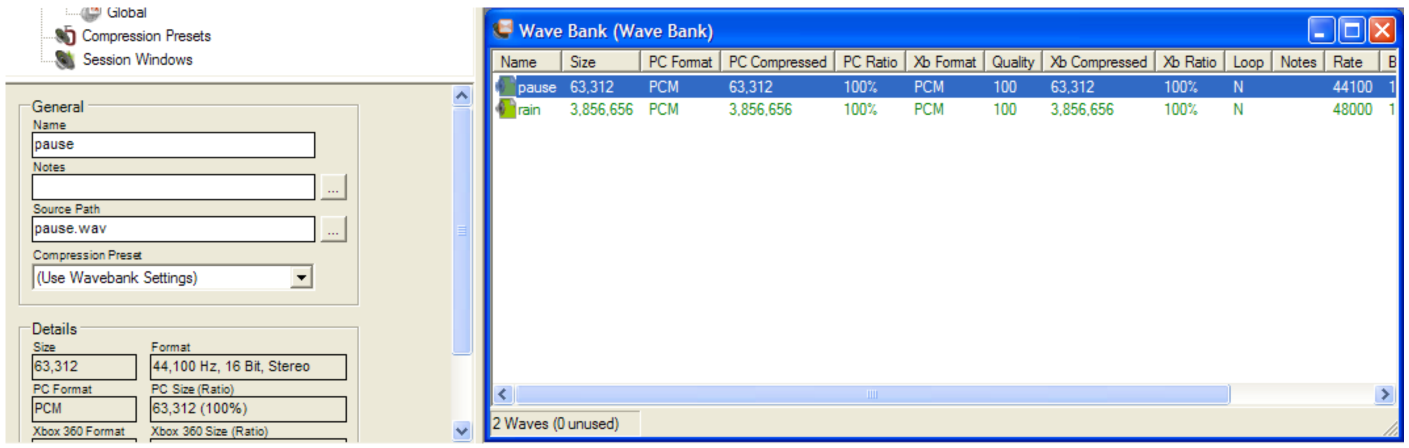


Figure 4 - New Sound Bank

New windows should appear in the grey space to the right (the 'work area'). To make these windows more managable, go to the tool bar and click Window > Tile Horizontally. Now that we've got the project to hold our sound items, we need to include them. You can find a ZIP file with our two sounds here:

http://www.insidegamer.org/documents/XNASounds.zip. Unzip the file and place the two audio files **into the Audio Backup directory**. Remember, we're trying to avoid awkward paths used for the files. Next, back in XACT, right click the Wave Bank window and select 'Insert Wave Files…'. Select the two sound files by browsing to the Audio Backup directory, selecting them both, and clicking Open. Your Wave Bank should now resemble:



General
Name: pause
Notes:
Source Path: pause.wav
Compression Preset: (Use Wavebank Settings)

Details
Size: 63,312
Format: 44,100 Hz, 16 Bit, Stereo
PC Format: PCM
PC Size (Ratio): 63,312 (100%)
Xbox 360 Format
Xbox 360 Size (Ratio)

Wave Bank (Wave Bank)

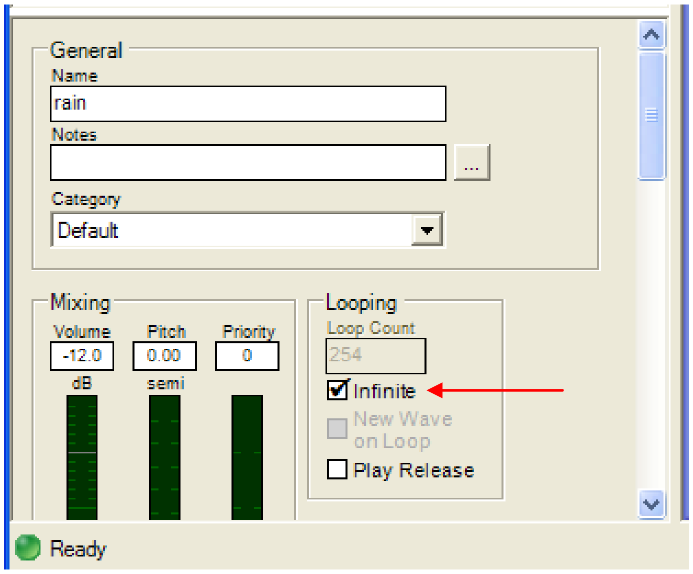
Name	Size	PC Format	PC Compressed	PC Ratio	Xb Format	Quality	Xb Compressed	Xb Ratio	Loop	Notes	Rate	B
pause	63,312	PCM	63,312	100%	PCM	100	63,312	100%	N		44100	1
rain	3,856,656	PCM	3,856,656	100%	PCM	100	3,856,656	100%	N		48000	1

2 Waves (0 unused)

Figure 5 - Wave Bank with 2 Sound Files

I've included the left portion of the screen for a reason. Notice the third textbox on the left is used for Source Path and reads 'pause.wav' (I have 'pause' selected). If you've included the sound from a different location, this will contain the path holding your sound file which will be what XNA uses to look for the file. Imagine giving your game to a friend and XNA tries looking a sound file in the path /My Documents/Homer/ and your friend for good reasons doesn't have that path. Suddenly the game will error and you'll be left with major fixes.

Now to create the sound cues. Highlight both sound files in the Wave Bank and drag them to the top portion of the Sound Bank. Placing them in the Sound Bank not only allows us to add cues, but also to modify the play back style of the sound. Click the 'rain' item in the Sound Bank and, in the Properties box (the box in the lower left hand corner of the screen), check the box next to 'Infinite'.


General
Name: rain
Notes:
Category: Default

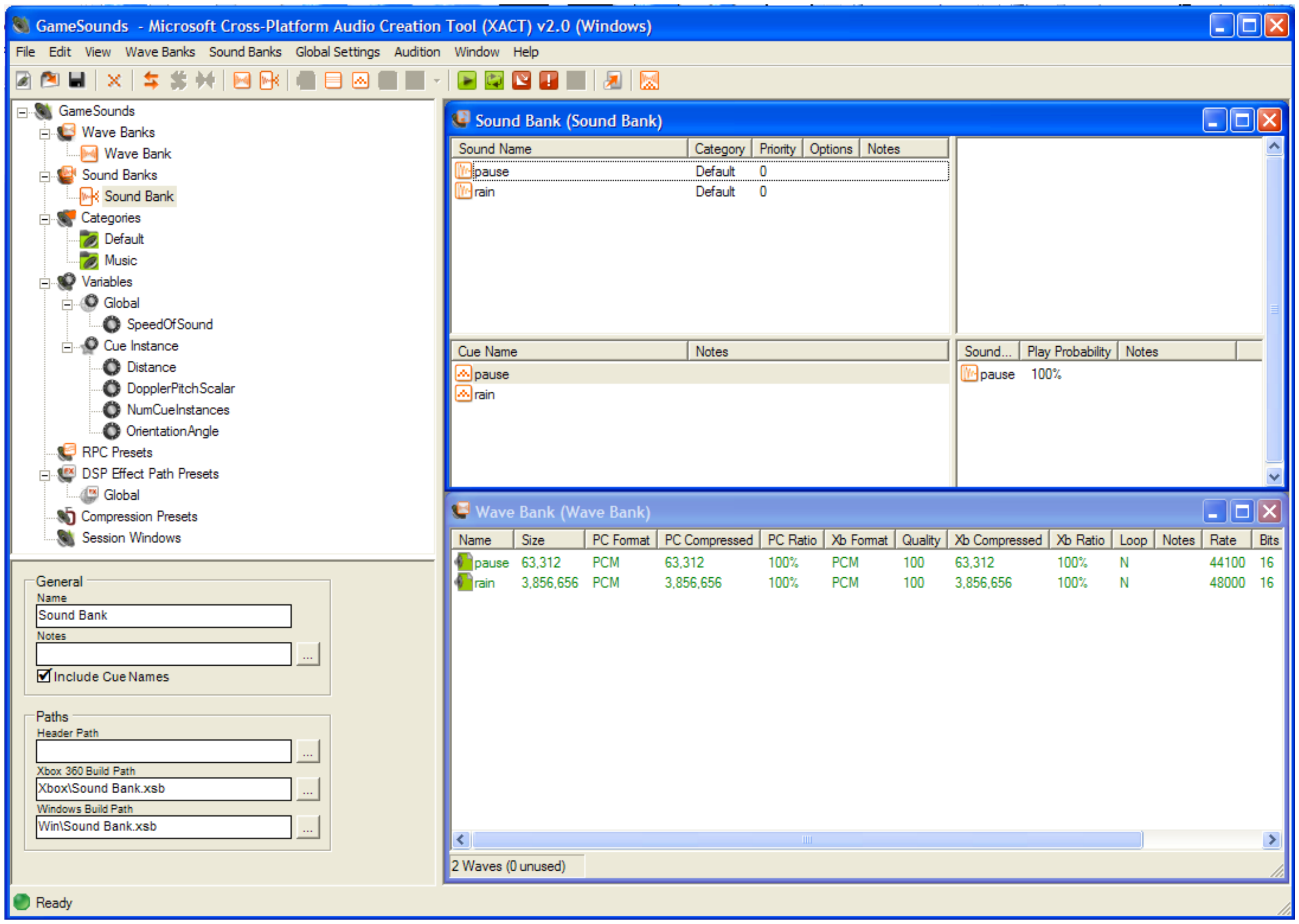
Mixing
Volume: -12.0 dB
Pitch: 0.00 semi
Priority: 0

Looping
Loop Count: 254
[✓] Infinite
[ ] New Wave on Loop
[ ] Play Release

Ready

Figure 6 - Properties of 'rain' (Infinite Checked)

Finally we are ready to build our cues. Return to the Sound Bank window, highlight 'pause' and 'rain' in the top half of the Sound Bank, and drag them into the bottom half under 'Cue Name'. This should automatically create two new cues named ‘pause’ and ‘rain’. Go to the toolbar and click ‘Save Project’. Your final XACT should look something like this:



Sound Name	Category	Priority	Options	Notes
pause	Default	0		
rain	Default	0		

Cue Name	Notes
pause	
rain	

Sound...	Play Probability	Notes
pause	100%	

Name	Size	PC Format	PC Compressed	PC Ratio	Xb Format	Quality	Xb Compressed	Xb Ratio	Loop	Notes	Rate	Bits
pause	63,312	PCM	63,312	100%	PCM	100	63,312	100%	N		44100	16
rain	3,856,656	PCM	3,856,656	100%	PCM	100	3,856,656	100%	N		48000	16

2 Waves (0 unused)

General
Name: Sound Bank
Notes:
☑ Include Cue Names

Paths
Header Path:
Xbox 360 Build Path: Xbox\Sound Bank.xsb
Windows Build Path: Win\Sound Bank.xsb

Figure 7 - XACT Final View

Next, we’ll import the XACT project before building the GameAudio class.

Importing XACT

While XACT is still fresh in your mind, lets import it into our XNA Project. As with the Sprites and Models, create a new folder in the Content subfolder called ‘Audio’ (remember, right click on ‘Content’ and select Add > New Folder). Right click on the Audio folder and select Add > Existing Item…, browse to the Audio Backup directory, and select only the XACT project file (GameSounds.xap).

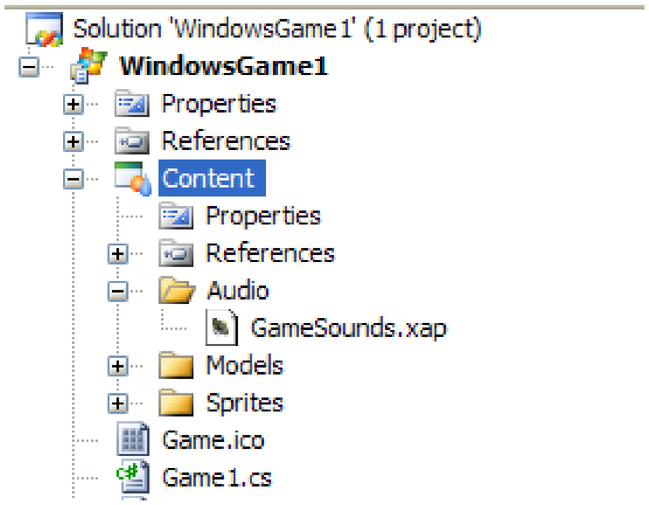


Figure 8 - Audio Folder

Using a new explorer window, copy the sound files 'rain.wav' and 'pause.wav' into the newly created Audio subfolder within your projects folder. At this point you should be able to compile and build your project without any errors. If you've accidentally copied the files into the wrong subfolder or have the Source Directory wrong in your XACT project, you will receive errors. The most common error mentions being unable to find the sound file which could be either one of these mistakes.

Writing the GameAudio Class

The GameAudio class is going to be the framework the Game1 class uses to access the sounds in the GameSounds project we've imported. Add a new class file and name it GameAudio.cs. Since we won't be using a lot of the XNA libraries with this class, we can simplify the Using Statements section as follows:

```
#region Using Statements
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
#endregion
```

We'll use a basic list like the GameSprite class which requires the System.Collections.Generic. Of course we want the basic XNA libraries including the audio library so we include the other items. Next, we need to initialize a bunch of global class variables. Place the following between your GameAudio class brackets:

```
//Audio Engine Items
private AudioEngine m_Engine;
private WaveBank m_WaveBank;
private SoundBank m_SoundBank;

//New Audio Struct
public struct audioItem
{
    public Cue audioCue;
    public bool isPlaying;
    public bool isPaused;
};

//List of game audio
public List<audioItem> m_gameAudio = new List<audioItem>();
```

XNA has three major requirements when linking your XNA game project with your XACT sound project, they are the first three variables. AudioEngine is the 'big boy' of the others and acts as the glue that holds the Sound Bank and Wave Bank together. The other two items specify the Wave Bank and Audio Bank items created by our XACT project respectively. Next, we create a new structure called audioItem which holds an audio cue and two booleans. The audioCue variable will hold the actual cue we'll work with, cues are what you call methods like Play() and Stop() on (think of it as holding the sound like the Texture2D in GameSprite held the sprite object). Next, we use booleans to see if the sound is playing or if it is paused, these are simple variables that are used later on to determine if we should stop the sound or unpause it or whatnot. Finally, we build a new list of the audioItem types to store all our sounds.

Note: In this class I've introduced yet another notation scheme (which is bad considering all other tutorials use other styles, but the exposure to this is more important). I've prefixed all the variables with 'm_*variable*' to specify that the variable is a member of the class. I could also, in Game1.cs, specify a variable as 'g_*variable*', this would mean it is a global variable.

GameAudio()

Now that the sound project is incorporated within our project and our sound files are in all the right places, we need to build the GameAudio class which will let us work with the sounds. Here's the GameAudio constructor:

```
/// <summary>
/// GameAudio constructor
/// </summary>
public GameAudio()
{
    m_Engine = new AudioEngine("Content\\Audio\\GameSounds.xgs");
    m_WaveBank = new WaveBank(m_Engine, "Content\\Audio\\Wave Bank.xwb");
    m_SoundBank = new SoundBank(m_Engine, "Content\\Audio\\Sound Bank.xsb");
}
```

The AudioEngine, WaveBank, and SoundBank objects are instantiated as new objects of whichever respective type they are declared as. The biggest confusion here is probably the crazy extension given in each item's constructor. Remember, the XACT project created a file as .xap, so where did these come from? Well, the XNA content pipeline, when the project is built, creates these files from the XACT project. I haven't mentioned this before, but all the content in your game gets built into a different format at build time. If you've looked into your /bin/x86/Debug/Content/ folder and into any of the subfolders, such as Sprites, you won't find .png files like we've included, but .xnb. XNA builds content like models and textures into .xnb files as well as it creates the .xgs, .xwb, and .xsb.

Note: Files cannot be added to an XNA project while the game is running for the reason XNA builds the file into its own format. Those files would need to be built into the respective XNA format (meaning a full project build) and then the game would need to be restarted with the new build.

~GameAudio()

The destructor for the GameAudio class is simple (as have been all the class destructors), we just get rid of the SoundBank, WaveBank, and AudioEngine objects.

```
/// <summary>
/// GameAudio destructor
/// </summary>
~GameAudio()
{
    m_SoundBank.Dispose();
    m_WaveBank.Dispose();
```

```
        m_Engine.Dispose();
    }
```

AddSound()

Before we can play the sound, we want to add it to the list of sounds able to be played. We'll do that with the AddSound() method:

```
    /// <summary>
    /// Adds a sound using sound filename
    /// </summary>
    /// <param name="sound"></param>
    public void AddSound(String sound)
    {
        audioItem newAudio;
        newAudio.audioCue = m_SoundBank.GetCue(sound);
        newAudio.isPaused = false;
        newAudio.isPlaying = false;
        m_gameAudio.Add(newAudio);
    }
```

Within the method we create a new audioItem to be added to the list. First we declare it as the audioItem type followed by setting the cue equal to a method called GetCue() which is part of the SoundBank object. Before, when we added cues to our project they defaulted to the same name as the file so what will happen, is we'll pass in the string 'rain' and the SoundBank object will look through all the cues for the one named 'rain' and if it's found will pull it out and set it equal to our new cue. If we specify a name that isn't there, we'd get a very nasty error (so don't do that!). Next, both booleans are set to false, we're just adding a file so it shouldn't be playing or paused. Finally we call the Add() method of our list. Most of this method (and class in general) should be very similar to the GameSprite class for you (creating new objects, adding them to the list, and later replacing them).

PlaySound()

The PlaySound() method will take a unique string, find the related audio item, and begin playback if it is found. Again, this should be pretty similar for the most part if you remember the GameSprite class. The search is exactly the same logic and code, but the action taken after the sound is found is different:

```
    /// <summary>
    /// Plays a sound based on sound filename
    /// </summary>
    /// <param name="sound"></param>
    public void PlaySound(String sound)
    {
        //Default variables
        int position = 0;
        bool found = false;

        //Look for sound in list
        for (int i = 0; i < m_gameAudio.Count; i++)
        {
            if (m_gameAudio[i].audioCue.Name == sound)
            {
                //Found sprite!
                found = true;
                position = i;
            }
        }
```

```
            //Can't update without an old sound object
            if (found)
            {
                audioItem localvar;
                localvar.isPaused = false;
                localvar.isPlaying = true;
                localvar.audioCue =
                    m_SoundBank.GetCue(m_gameAudio[position].audioCue.Name);

                m_gameAudio[position] = localvar;
                m_gameAudio[position].audioCue.Play();
            }
        }
```

I'll skip right to the 'if (found)' section because you should understand the top by now. After the sound object is found, we're going to make a new local sound object and swap it in. Obviously, the sound is playing so isPaused is set to false and isPlaying is set to true. Then, the audio cue (remember, that's simply the actual audio file) is set equal to the old sound object's audio cue. We can't just set the two equal though, we need to find that cue as it's stored in the SoundBank, so we use the GetCue() method of the SoundBank and tell it to find the old cue. Once it's found, it's returned to our local object. Next, the audio item in the list is swapped out with the local version (similar to the GameSprite class). Finally, we tell the audio cue to begin playing simply by calling Play() of the cue type. At this point, the sound should begin playing.

PauseSound()

PauseSound() does almost the exact same as the PlaySound() method but (obviously), pauses the sound. The code:

```
        /// <summary>
        /// Pauses specified sound
        /// </summary>
        /// <param name="sound"></param>
        public void PauseSound(String sound)
        {
            //Default variables
            int position = 0;
            bool found = false;

            //Look for sound in list
            for (int i = 0; i < m_gameAudio.Count; i++)
            {
                if (m_gameAudio[i].audioCue.Name == sound)
                {
                    //Found sprite!
                    found = true;
                    position = i;
                }
            }

            //Can't update without an old sound object
            if (found && m_gameAudio[position].isPlaying)
            {
                audioItem localvar;
                localvar.isPaused = true;
                localvar.isPlaying = false;
                localvar.audioCue = m_gameAudio[position].audioCue;

                m_gameAudio[position] = localvar;

                m_gameAudio[position].audioCue.Pause();
        }
    }
```

Again, the sound is searched for and will be replaced by a local item. Notice the if statement before the update code now checks to see if the sound is playing. There's no need to pause a sound that isn't playing so we can first check if the isPlaying boolean is true and skip this method call if it is. Now, though, the item is paused so isPaused is set to true and isPlaying is set to false. The next two lines are the same and is followed by the line that actually pauses the cue with the Pause() method.

UnpauseSound()

You should begin to notice a pattern with the code. All the code to access the sound item is the same while one or two method changes make the sound pause, play, or in this case, resume from a paused state.

```
    /// <summary>
    /// Resumes specified sound
    /// </summary>
    /// <param name="sound"></param>
    public void UnpauseSound(String sound)
    {
        //Default variables
        int position = 0;
        bool found = false;

        //Look for sound in list
        for (int i = 0; i < m_gameAudio.Count; i++)
        {
            if (m_gameAudio[i].audioCue.Name == sound)
            {
                //Found sprite!
                found = true;
                position = i;
            }
        }

        //Can't update without an old sound object
        if (found && m_gameAudio[position].isPaused)
        {
            audioItem localvar;
            localvar.isPaused = false;
            localvar.isPlaying = true;
            localvar.audioCue = m_gameAudio[position].audioCue;

            m_gameAudio[position] = localvar;

            m_gameAudio[position].audioCue.Resume();
        }
    }
```

This time, the if statement checks if the item is paused because if the sound item isn't paused, 'resuming' or 'unpausing' would be quite useless. The sound will resume playing so isPaused is set to false and isPlaying to true. Finally, we call Resume() of the cue type to resume the paused sound.

StopSound()

Once a sound is playing we have the choice to pause or stop it. We've covered pause, now stopping:

```
    /// <summary>
    /// Stops specified sound
```

```
        /// </summary>
        /// <param name="sound"></param>
        public void StopSound(String sound)
        {
            //Default variables
            int position = 0;
            bool found = false;

            //Look for sound in list
            for (int i = 0; i < m_gameAudio.Count; i++)
            {
                if (m_gameAudio[i].audioCue.Name == sound)
                {
                    //Found sprite!
                    found = true;
                    position = i;
                }
            }

            //Can't update without an old sound object
            if (found && m_gameAudio[position].isPlaying)
            {
                audioItem localvar;
                localvar.isPaused = false;
                localvar.isPlaying = false;
                localvar.audioCue = m_gameAudio[position].audioCue;

                m_gameAudio[position] = localvar;

                m_gameAudio[position].audioCue.Stop(AudioStopOptions.Immediate);
            }
        }
```

The if statement checks if the sound object is currently playing before stopping it. The object should be stopped after this so isPaused and isPlaying are both set to false. The swap happens as normal and is followed by the line used to stop the sound but it looks different than the others. Stop() requires a parameter of the AudioStopOptions type which can be set to Immediate or AsAuthored. Immediate stops the sound where it currently is at while AsAuthored plays any transition specified in our XACT project. For our current uses, Immediate should suffice.

StopAll()

StopAll() is a simple method used to stop every sound that is currently playing. While I won't use this method at this current time, it's very useful when transitioning from game states such as from MainMenu to InGame.

```
        /// <summary>
        /// Stops all playing/paused sounds
        /// </summary>
        public void StopAll()
        {
            for (int i = 0; i < m_gameAudio.Count; i++)
            {
                if (m_gameAudio[i].isPlaying || m_gameAudio[i].isPaused)
                {
                    audioItem localvar;
                    localvar.isPaused = false;
                    localvar.isPlaying = false;
                    localvar.audioCue = m_gameAudio[i].audioCue;

                    m_gameAudio[i] = localvar;
                    m_gameAudio[i].audioCue.Stop(AudioStopOptions.Immediate);
                }
            }
        }
```

A simple for loop runs through each audio object in the audio list and if the sound is playing or paused, we swap that item out with a new item that is stopped (similar to StopSound()).

PauseAll()

PauseAll() is programatically almost identical to the StopAll() method though pauses all the objects that are playing which can be useful between InGame and Pause states where all the sounds need to be paused but resumed after being paused. It is unused in this example because we only have two sounds but in bigger projects would be very useful:

```
    /// <summary>
    /// Pauses all playing sounds
    /// </summary>
    public void PauseAll()
    {
        for (int i = 0; i < m_gameAudio.Count; i++)
        {
            if (m_gameAudio[i].isPlaying)
            {
                audioItem localvar;
                localvar.isPaused = true;
                localvar.isPlaying = false;
                localvar.audioCue = m_gameAudio[i].audioCue;

                m_gameAudio[i] = localvar;

                m_gameAudio[i].audioCue.Pause();
            }
        }
    }
```

UnpauseAll()

Generally same as StopAll() and PauseAll() with the exception that it unpauses any paused sound. Useful when moving between Pause and InGame states:

```
    /// <summary>
    /// Unpauses all paused sounds
    /// </summary>
    public void UnpauseAll()
    {
        for (int i = 0; i < m_gameAudio.Count; i++)
        {
            if (m_gameAudio[i].isPaused)
            {
                audioItem localvar;
                localvar.isPaused = false;
                localvar.isPlaying = true;
                localvar.audioCue = m_gameAudio[i].audioCue;

                m_gameAudio[i] = localvar;

                m_gameAudio[i].audioCue.Resume();
            }
        }
    }
```

UpdateAudio()

UpdateAudio() is the other incredibly important method in the GameAudio class. Just like the video card needs to call Draw() each frame to draw any objects, the sound system needs to be updated every frame. XNA easily accomodates this and includes an Update() method for the AudioEngine which we simply call:

```
    /// <summary>
    /// Updates Sound Engine
    /// </summary>
    public void UpdateAudio()
    {
        m_Engine.Update();
    }
```

Implementing the GameAudio Class

Quickly before continuing, lets review what will happen with our demo. While at the InGame state, the sound of rain falling will continuously loop in the background. When pause is pressed, the rain wil pause and a bell will sound. Once pause is pressed a second time, the bell resounds and the rain resumes.

First, add another object to the top of your Game1 class and define it as type GameAudio, such as:

```
    GameAudio gameAudio;
```

Next, in your LoadContent() method, after initializing your GameSprite object initialize your GameAudio object with the following:

```
        gameAudio = new GameAudio();
```

This should be located just above a bunch of lines that are importing your sprite objects. After the sprite objects, add the following before your state setting:

```
        //Load Audio
        gameAudio.AddSound("rain");
        gameAudio.AddSound("pause");
```

In the constructor of the GameAudio class we already specified the XACT project to use so all we need to do is add our sounds and work with them. These lines add the rain and pause sounds (be careful, they're case-sensitive). Then, either after or before the state setting (preferably after), add this line to play the rain sound (since we'll be InGame):

```
        gameAudio.PlaySound("rain");
```

That was a lot to add and was all over, here's what your LoadContent() method should generally resemble:

```
    /// <summary>
    /// LoadContent will be called once per game and is the place to load
    /// all of your content.
    /// </summary>
    protected override void LoadContent()
    {
        gameStates.ResetStates();
        gameStates.Loading = true;

        // Create a new SpriteBatch, which can be used to draw textures.
        spriteBatch = new SpriteBatch(GraphicsDevice);
```

```
            gameSprites = new GameSprite(this.Content, spriteBatch);

            gameAudio = new GameAudio();


            gameSprites.AddSprite("RadarBG", "RadarBG",
                  graphics.GraphicsDevice.Viewport.Width - 100, 100, 0.0f, 0.7f, true);

            gameSprites.AddSprite("RadarChar", "RadarChar",
                  graphics.GraphicsDevice.Viewport.Width - 100, 100, 0.0f, 0.7f, true);

            gameSprites.AddSprite("4Buttons", "4Buttons",
                  graphics.GraphicsDevice.Viewport.Width - 90,
                  graphics.GraphicsDevice.Viewport.Height - 90, 0.0f, 1.0f, true);

            gameSprites.AddSprite("HealthBG", "HealthBG", 140,
                  graphics.GraphicsDevice.Viewport.Height - 100, 0.0f, 1.0f, true);

            gameSprites.AddSprite("HealthBorder", "HealthBorder", 140,
                  graphics.GraphicsDevice.Viewport.Height - 100, 0.0f, 1.0f, true);

            //Load Pause Screen
            gameSprites.AddSprite("PauseBG", "PauseBG",
                  graphics.GraphicsDevice.Viewport.Width/2,
                  graphics.GraphicsDevice.Viewport.Height/2, 0.0f, 1.0f, false);

            gameSprites.AddSprite("PauseMenuLargeBG", "PauseMenuLargeBG",
                  graphics.GraphicsDevice.Viewport.Width/2,
                  graphics.GraphicsDevice.Viewport.Height/2, 0.0f, 1.0f, false);

            gameSprites.AddSprite("PauseMenuExit", "PauseMenuExit",
                  graphics.GraphicsDevice.Viewport.Width / 2,
                  graphics.GraphicsDevice.Viewport.Height / 2 + 200, 0.0f, 1.0f, false);

            gameSprites.AddSprite("PauseMenuExitSelected", "PauseMenuExitSelected",
                  graphics.GraphicsDevice.Viewport.Width / 2,
                  graphics.GraphicsDevice.Viewport.Height / 2 + 200, 0.0f, 1.0f, false);

            //Load Audio
            gameAudio.AddSound("rain");
            gameAudio.AddSound("pause");

            gameStates.ResetStates();
            gameStates.InGame = true;

            gameAudio.PlaySound("rain");
        }
```

Note: The recent additions are bolded.

Update()

So the sounds have been added to the list, next we need to change what happens when pause is pressed. Locate the Update() method in your Game1.cs file. Towards the end of the if which checks if InGame is currently true, there is a check to see if the spacebar (pause button) has been pressed. Update this code to resemble the following:

```
                //Pause Game
                if (Keyboard.GetState().IsKeyDown(Keys.Space) && (remKey._key !=
                     Keys.Space && remKey._keyPressed == false))
                {
```

```
                remKey._key = Keys.Space;
                remKey._keyPressed = true;

                gameAudio.PauseSound("rain");
                gameAudio.PlaySound("pause");

                ShowPauseMenu();
                this.IsMouseVisible = true;

                gameStates.ResetStates();
                gameStates.Paused = true;
            }
```

Note: The new lines are bolded.

If the pause button is pressed, we pause the rain sound and play the pause sound. Since we didn't specify loop for the pause sound in our XACT project, the pause sound should only play once.

Next, we need to update when the player presses pause to resume the game. This code should be located only a few lines below what you've just updated and should resemble:

```
        if (gameStates.Paused)
        {
            CheckPauseMenu();

            if (Keyboard.GetState().IsKeyDown(Keys.Space) && (remKey._key !=
                Keys.Space && remKey._keyPressed == false))
            {
                remKey._key = Keys.Space;
                remKey._keyPressed = true;

                gameAudio.PlaySound("pause");
                gameAudio.UnpauseSound("rain");

                HidePauseMenu();
                this.IsMouseVisible = false;

                gameStates.ResetStates();
                gameStates.InGame = true;
            }
        }
```

Note: The new lines are bolded.

Again, we simply play the pause sound a second time and unpause the rain sound.

Immediately after this (still within the Update() method), add this line:

```
        gameAudio.UpdateAudio();
```

Note: This should appear **before** the base.Update(gameTime) line!

This line simply updates all our audio as described before.

Conclusion

This was definitely a complicated tutorial, especially considering we needed to create a new project in XACT, write the GameAudio class, and implement it (implementation was the simplest part!). It should be pretty obvious that the XACT tool really helps slim down the code used in the actual game files and allows you to specify audio changes without having to code them all. If you've had troubles with this tutorial, simply step through it a few more times or read over the area you had trouble with. There's a good chance many readers will accidentally mix up some portion of the XACT project so don't be afraid to start your XACT project over.

Suggested Exercises

1) Add two more sounds that play at specific times (remember, XNA only supports Wave files).
2) Substitute one of your Pause() calls with PauseAll().
3) Why should a method like PauseAll() come before any subsequent Play() method calls?
4) Currently StopSound() uses the Immediate setting to stop the sound. Pseudo-code a stop method that allows the caller to specify to either use Immediate or AsAuthored. [Pseudo-code means write a general overview of a method as you would speak it to someone.]
5) Play some video game and list everytime you notice a sound play as the game state changes.