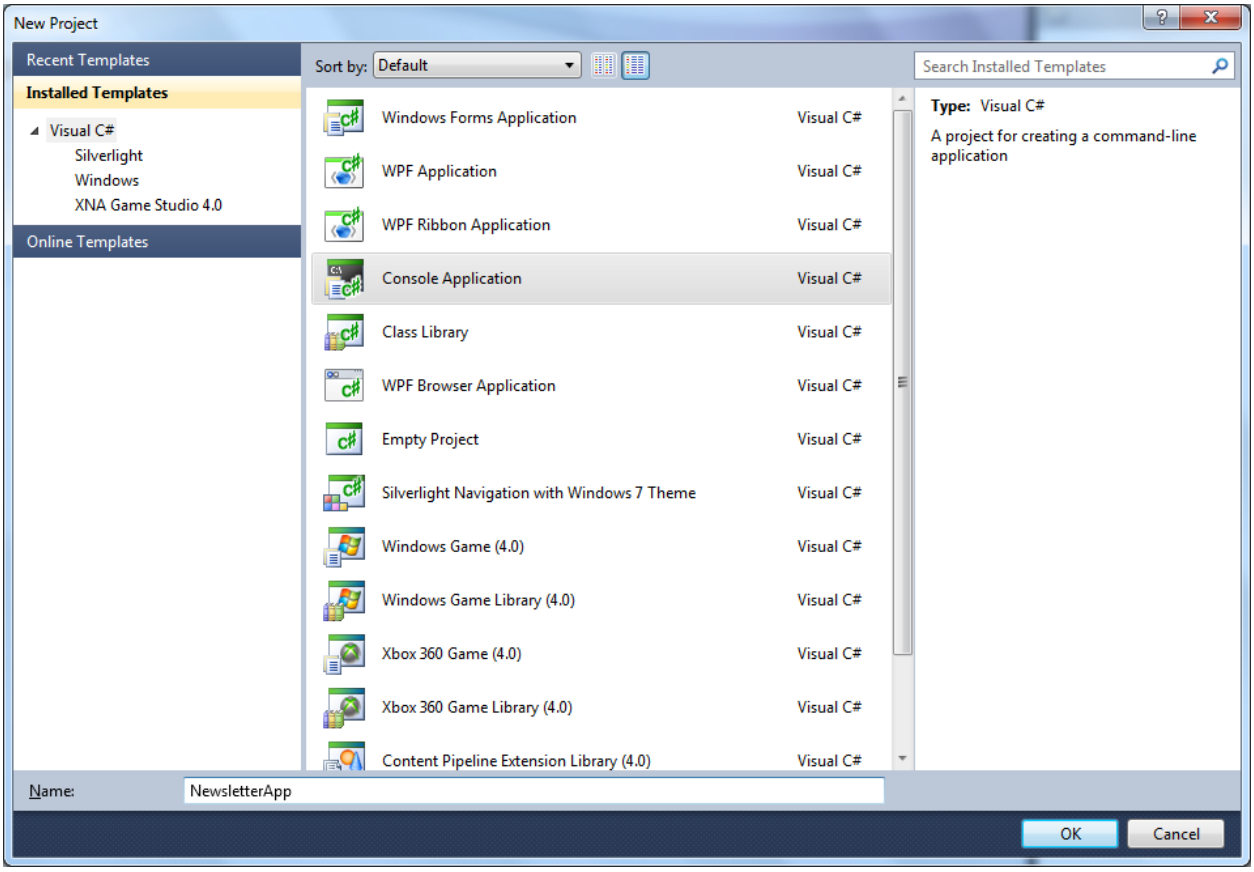
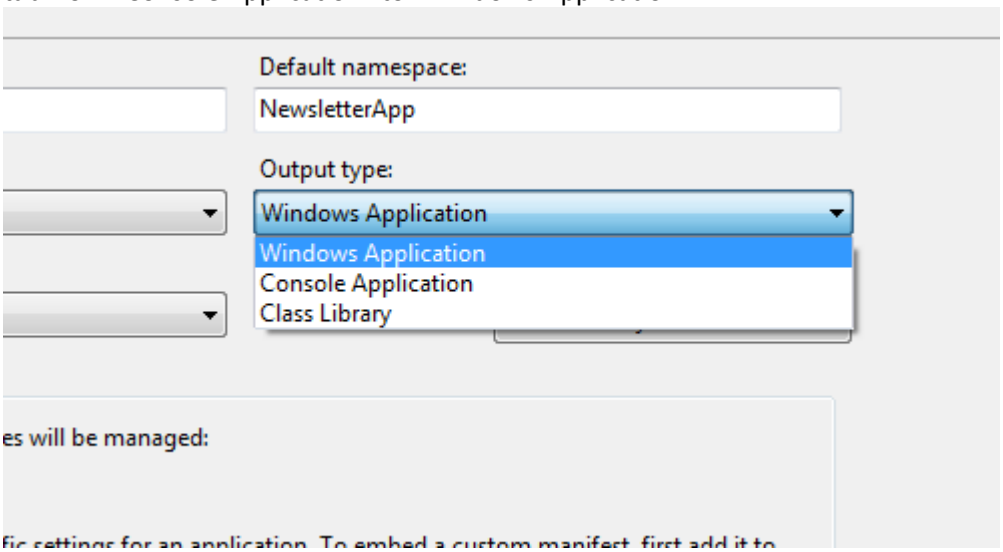
If you are a fan of the .NET family of languages – C#, Visual Basic, and so forth – and you own a copy of AGK, then you've got a new toy to play with.
The AGK Wrapper for .NET is an open source project that provides a coupling between the AGK library and the Common Language Runtime (the source can be found at http://agkwrapper.codeplex.com/, the compiled binaries are available from your order history). Now, before you get too excited, it should be noted that as-is the wrapper is Windows only, for a variety of reasons that I won't get into here. So if you are planning on making the next cross-platform blockbuster, you may want to get dirty with C++ or stick to Tier 1 Basic. But if your target is Windows, or you want the rapid prototyping power of C#/VB, then read on.
For this example I'll be using C#, but most of what I'm doing applies to VB as well (and I'll throw some VB specific notes in as well).

Getting Started

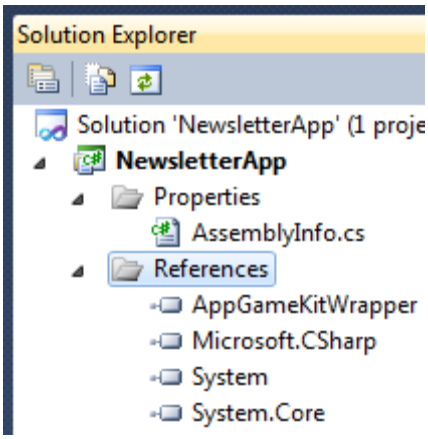
First, you need to create new project. I'm going to start with a Console project, because it is a nice minimal setup with few dependencies and only a couple settings that need to be adjusted.



Once we've created the console project, we need to change a few things. For one, we need to change it from a console application to a normal Windows application (since I assume you don't want a command line popping up with your game window!). Open your newly created project's properties page (right click the project name and select "properties") and change the "Output Type" field in the "Application" tab from "Console Application" to "Windows Application".



Next we need to add the wrapper as a dependency. Right click on the "References" folder in your project tree view, and select "Add Reference". Browse to wherever you have installed/saved the AGK Wrapper and select it.



You are pretty much ready to go, but if you run the project now, you'll find that nothing pops up. The "Main" method in the template generated "Program.cs" file is executed, but nothing is in it!
Now you might think that, like Tier 1, you could just jump in and start adding AGK commands here. Unfortunately it isn't so simple – you haven't got a window yet! If you've used AGK in Tier 2, you'll know that your game sits inside of an "app" class with "Begin", "Loop", and "End" methods. The wrapper provides an abstract base class for this, aptly named "App". You'll find it in the "TheGameCreators.AppGameKit" namespace. You should create a new class to derive from this – I called it "Game".

The first thing you should do is create a constructor that sets some of the initial settings. Most of these properties won't do anything after the game begins running, you'll have to use the relevant AGK commands. Particularly, you should set your device width and height. It's probably not a bad idea to center it either.

```
public class Game : TheGameCreators.AppGameKit.App
{
    public Game()
    {
        // Setup your window size
        DeviceWidth = 1024;
        DeviceHeight = 768;

        CenterDevice();
    }
```

There are a few properties you may not recognize that appear in the intellisense:

```
    // if false, sleeps the game to conserve processing power
    AllowKeepActive = true;

    // allows more than one game window to be opened
    AllowMultipleInstances = true;
}
```

These are settings that are available in Tier 2 but that may not be particularly straightforward. Generally you shouldn't need to worry about them, as the defaults will usually suffice. However, you will probably want to set the Name property:

```
Name = "The Newsletter Game";
```

Once you've done that, you will want to override the Begin, Loop, and End methods. You'll find that, because they are marked as abstract, you *must* do this (but you should want to anyways!).

```
protected override void Begin()
{

}

protected override void Loop()
{

}

protected override void End()
{
```

If you haven't already, you'll want to import the wrapper's namespace:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TheGameCreators.AppGameKit;
```

Now you can start using the "Agk" class! Modify your loop to look like this:

```
protected override void Loop()
{
    Agk.Print("Hello, .NET Framework!");
    Agk.Sync();
}
```

Now, if you are using VB, things are a little different. Because AGK already uses the namespace "AGK" in the C++ world, it is – unfortunately – exported along with everything else in the wrapper, despite containing only unmanaged (non-CLR) types. VB isn't case sensitive, so it can't distinguish between the "Agk" static class and the "AGK" namespace. There is an easy workaround however:

```
REM Fortunately autocomplete makes this alternative relatively painless to use
Imports AgkNet = TheGameCreators.AppGameKit.Agk
```

Since Visual Studio's autocomplete is very aggressive, it is pretty easy to work with the longer name – just let it fill it in for you. This works best if you don't import the entire namespace, but only the static class. You can use any alias you like of course; you don't *have* to use "AgkNet". Also, since Loop and End are keywords, you need to wrap them in brackets (VS will do this for you, if you put them in by typing "Overrides" and using auto complete):

```
Protected Overrides Sub [End]()
```

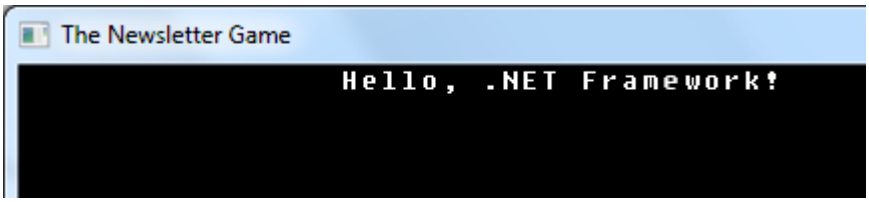
So, hit *Start Debugging* and you are ready to go, right? Well, not quite. There is nothing that creates an instance of our game class! Go back to the "Program" class created by our template and fill in this:

```
static void Main(string[] args)
{
    Game game = new Game();

    int errorCode = game.Run();

    if (errorCode != 0)
        throw new Exception("Game exited with non-zero code!");
}
```

This creates an instance of our game, and calls its “Run” method (which is blocking, by the way – it won’t return until the game exits). The game can also return an error code – zero means the game ran fine; anything other than zero is an error. At this time the error codes are not specific, and are most likely to occur only if “AllowMultipleInstances” is set to false and a second copy of the game is started.
Now our game is ready – hit compile and run, and you should get a game window:



From here, you can get to working on your next great blockbuster. The Space Shooter example is ported (almost verbatim) to C# and Visual Basic and is included with the AGK Wrapper itself, and is also available with the source code’s documentation at http://agkwrapper.codeplex.com/documentation.