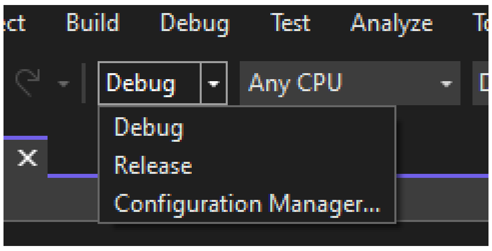
48. What is the difference between Debug and Release builds?

Brief summary: During the Release build, the compiler applies optimizations it finds appropriate. Because of that, the result of the build is often smaller and it works faster. On the other hand, it's harder to debug because the compiled result doesn't match the source code exactly.

When using Visual Studio, it's hard not to notice this little option in the top menu:

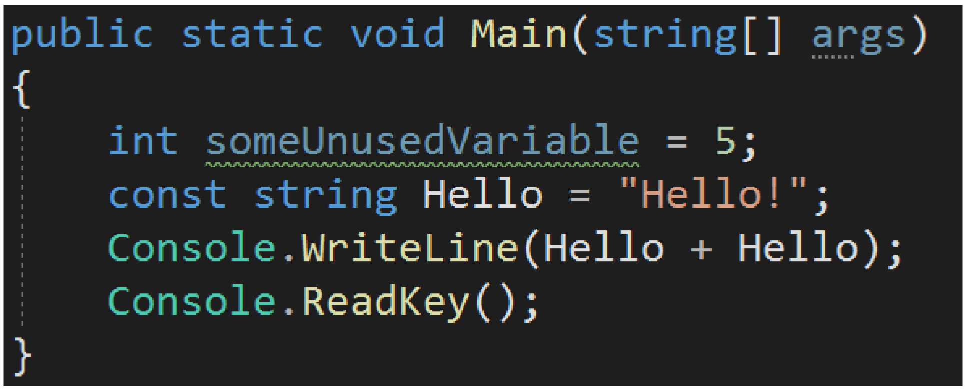


When developing the code, we most likely use the Debug mode and don't really think much about the other option - the Release mode. As their names suggest, the Debug mode is most appropriate when debugging the application, so mostly during the development, and the Release mode should be used when we intend to release the application so it can be used by users or other programs.

But what is the difference between them, exactly?

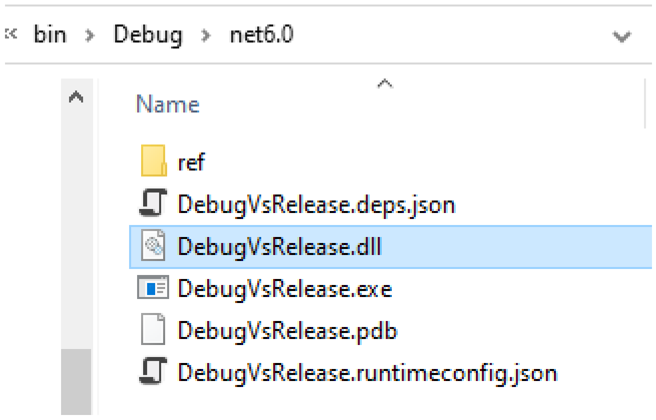
The main difference is that in the Release build, the compiler introduces some optimizations it finds appropriate. They can be things like removing unused variables or simplifying method calls. It helps to make the final CIL code smaller, simpler, and often faster. Remember, CIL stands for the Common Intermediate Language, which is the language to which the compiler compiles C# code. Those optimizations may make debugging harder because the compiled result doesn't match exactly the source code.

Let's see the simple compiler optimizations in practice. First, let me show you some code that is not optimal:



```
public static void Main(string[] args)
{
    int someUnusedVariable = 5;
    const string Hello = "Hello!";
    Console.WriteLine(Hello + Hello);
    Console.ReadKey();
}
```

As you can see, we have an unused variable here. Let me build this project in **Debug** mode. The build output will end up in the Debug folder:



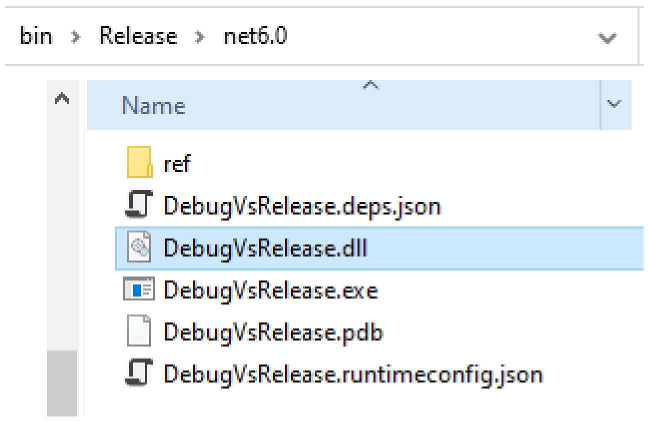
```
« bin › Debug › net6.0

Name
ref
DebugVsRelease.deps.json
DebugVsRelease.dll
DebugVsRelease.exe
DebugVsRelease.pdb
DebugVsRelease.runtimeconfig.json
```

Here we can see the dll being the result of the program compilation. What I'm going to do now is to first see the CIL code using **ildasm**, and then I will use SharpLab.io to translate it back to C#, to see how the compilation affected the structure of the code:

```
public class Program
{
    [System.Runtime.CompilerServices.NullableContext(1)]
    public static void Main(string[] args)
    {
        int num = 5;
        Console.WriteLine("Hello!Hello!");
        Console.ReadKey();
    }
}
```

Well, it's not very exciting. We can see that the unused variable has been renamed to something shorter and that the const strings have been inlined. Now, let's do the same thing but in the **Release** build. First of all, we will see it ends up in a different folder:



```
ref
DebugVsRelease.deps.json
DebugVsRelease.dll
DebugVsRelease.exe
DebugVsRelease.pdb
DebugVsRelease.runtimeconfig.json
```

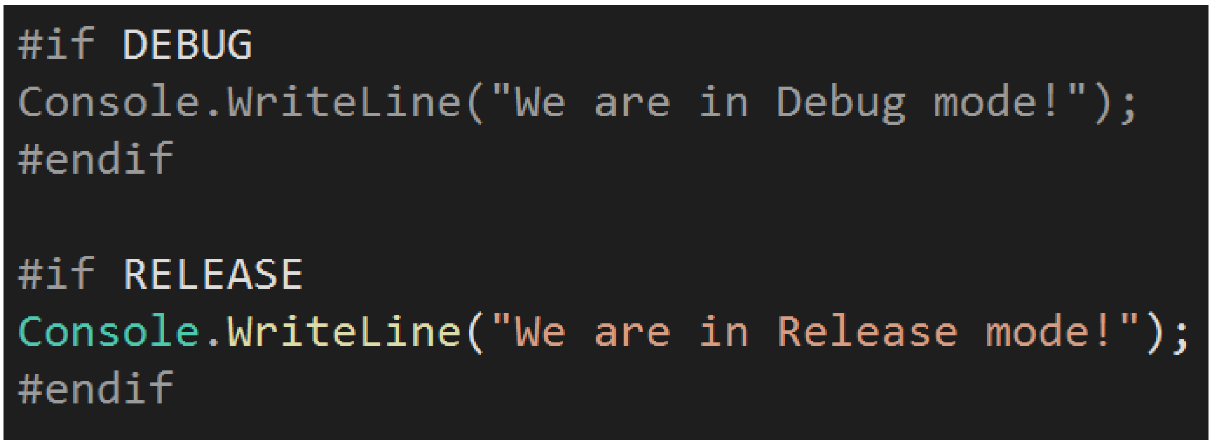
As you can see, the Release folder is used now. I will again take the CIL code that has been created by the compiler and will paste it to SharpLab to see what C# it translates to:

```
public class Program
{
    [System.Runtime.CompilerServices.NullableContext(1)]
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello!Hello!");
        Console.ReadKey();
    }
}
```

This time, the unused variable has been removed completely.

This was just an extremely simple example of what the compiler can optimize in the Release mode, but be aware that those optimizations can be much more serious. The compiler can simplify or inline method calls, remove whole pieces of code that it doesn't find useful, and so on. The algorithm behind that is quite complex and it depends on the compiler version, as well as such low-level details like how big the methods are. We can configure some of the optimizations settings with attributes like MethodImplOptions.NoInlining. You can read more about it here:
https://docs.microsoft.com/en-us/dotnet/api/system.runtime.compilerservices.methodimploptions?view=net-6.0

The code optimization done by the compiler is the main difference between the Debug and Release modes. Please note that if we want some code to be executed only in one of those modes, we can put it in **#if DEBUG** or **#if RELEASE** conditional preprocessor directives:



```
#if DEBUG
Console.WriteLine("We are in Debug mode!");
#endif

#if RELEASE
Console.WriteLine("We are in Release mode!");
#endif
```

Let’s summarize. During the Release build, the compiler applies optimizations it finds appropriate. Because of that, the result of the build is often smaller and it works faster. On the other hand, it’s harder to debug because the compiled result doesn’t match the source code exactly.

Bonus questions:

- "**How can we execute some piece of code only in the Debug, or only in the Release mode?**"
  *By placing it inside a **#if DEBUG** or **#if RELEASE** conditional preprocessor directives.*