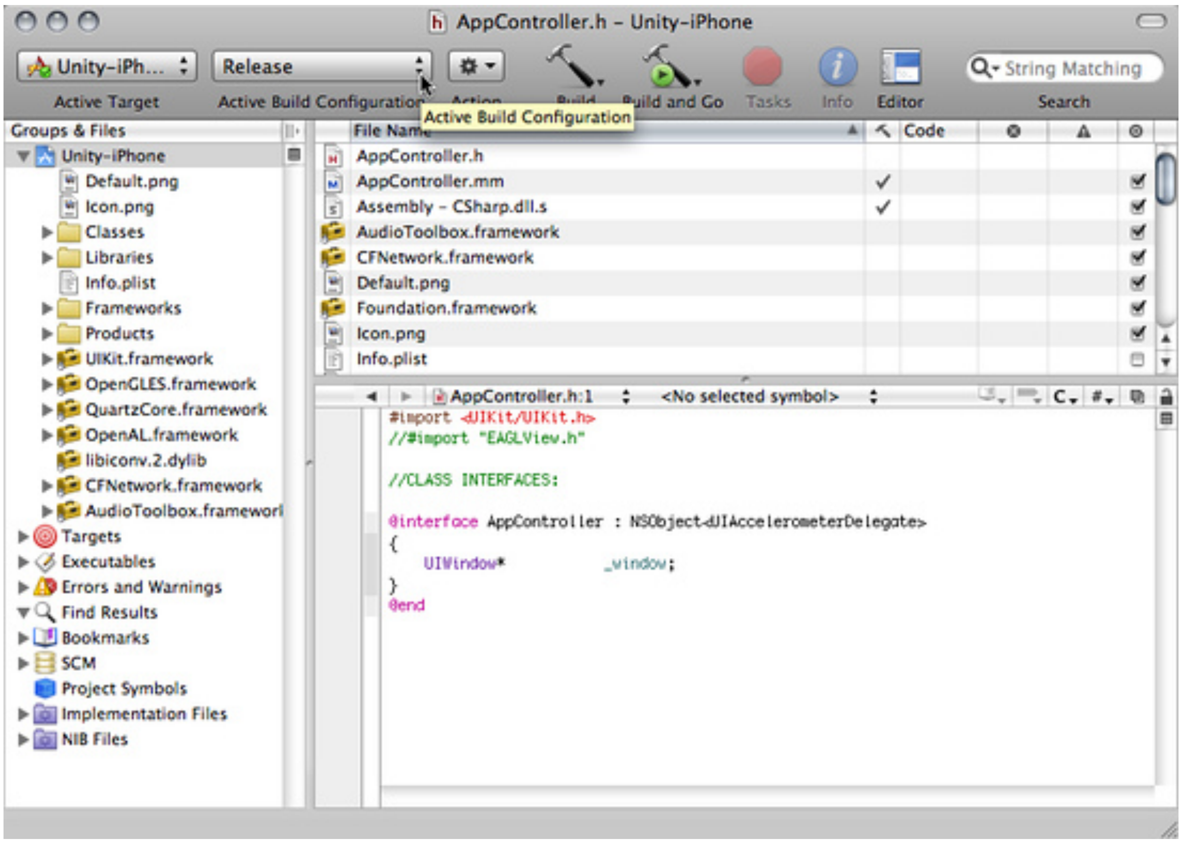
Optimizing the Size of the Built iOS Player

The two main ways of reducing the size of the player are by changing the **Active Build Configuration** within Xcode and by changing the **Stripping Level** within Unity.

Building in Release Mode

You can choose between the **Debug** and **Release** options on the **Active Build Configuration** drop-down menu in Xcode. Building as **Release** instead of **Debug** can reduce the size of the built player by as much as 2-3MB, depending on the game.



```
#import <UIKit/UIKit.h>
//#import "EAGLView.h"

//CLASS INTERFACES:

@interface AppController : NSObject<UIAccelerometerDelegate>
{
    UIWindow*           _window;
}
@end
```

The Active Build Configuration drop-down

In Release mode, the player will be built without any debug information, so if your game crashes or has other problems there will be no stack trace information available for output. This is fine for deploying a finished game but you will probably want to use Debug mode during development.

iOS Stripping Level (Advanced License feature)

The size optimizations activated by stripping work in the following way:-

1. **Strip assemblies** level: the scripts' bytecode is analyzed so that classes and methods that are not referenced from the scripts can be removed from the DLLs and thereby excluded from the AOT compilation phase. This optimization reduces the size of the main binary and accompanying DLLs and is safe as long as no reflection is used.

2. **Strip ByteCode** level: any .NET DLLs (stored in the Data folder) are stripped down to metadata only. This is possible because all the code is already precompiled during the AOT phase and linked into the main binary.

3. **Use micro mscorlib** level: a special, smaller version of mscorlib is used. Some components are removed from this library, for example, Security, Reflection.Emit, Remoting, non Gregorian calendars, etc. Also, interdependencies between internal components are minimized. This optimization reduces the main binary and mscorlib.dll size but it is not compatible with some System and System.Xml assembly classes, so use it with care.

These levels are cumulative, so level 3 optimization implicitly includes levels 2 and 1, while level 2 optimization includes level 1.

Note: Micro mscorlib is a heavily stripped-down version of the core library. Only those items that are required by the Mono runtime in Unity remain. Best practice for using micro mscorlib is not to use any classes or other features of .NET that are not required by your application. GUIDs are a good example of something you could omit; they can easily be replaced with custom made pseudo GUIDs and doing this would result in better performance and app size.

Tips

How to Deal with Stripping when Using Reflection

Stripping depends highly on static code analysis and sometimes this can't be done effectively, especially when dynamic features like reflection are used. In such cases, it is necessary to give some hints as to which classes shouldn't be touched. Unity supports a per-project custom stripping *blacklist*. Using the blacklist is a simple matter of creating a **link.xml** file and placing it into the **Assets** folder. An example of the contents of the **link.xml** file follows. Classes marked for preservation will not be affected by stripping:-

```
<linker>
       <assembly fullname="System.Web.Services">
               <type fullname="System.Web.Services.Protocols.-
SoapTypeStubInfo" preserve="all"/>
               <type fullname="System.Web.Services.Configura-
tion.WebServicesConfigurationSectionHandler" preserve="all"/>
       </assembly>
```

```
        <assembly fullname="System">
                <type fullname="System.Net.Configuration.WebRe-
questModuleHandler" preserve="all"/>
                <type fullname="System.Net.HttpRequestCreator"
preserve="all"/>
                <type fullname="System.Net.FileWebRequestCre-
ator" preserve="all"/>
        </assembly>
</linker>
```

Note: it can sometimes be difficult to determine which classes are getting stripped in error even though the application requires them. You can often get useful information about this by running the stripped application on the simulator and checking the Xcode console for error messages.

Simple Checklist for Making Your Distribution as Small as Possible

1. Minimize your assets: enable PVRTC compression for textures and reduce their resolution as far as possible. Also, minimize the number of uncompressed sounds. There are some additional tips for file size reduction here.
2. Set the iOS Stripping Level to **Use micro mscorlib**.
3. Set the script call optimization level to **Fast but no exceptions**.
4. Don't use anything that lives in System.dll or System.Xml.dll in your code. These libraries are **not** compatible with micro mscorlib.
5. Remove unnecessary code dependencies.
6. Set the API Compatibility Level to **.Net 2.0 subset**. Note that .Net 2.0 subset has limited compatibility with other libraries.
7. Set the Target Platform to **armv6 (OpenGL ES1.1)**.
8. Don't use JS Arrays.
9. Avoid generic containers in combination with value types, including structs.

Can I produce apps of less than 20 megabytes with Unity?

Yes. An empty project would take about 13 MB in the AppStore if all the size optimizations were turned off. This gives you a budget of about 7MB for compressed assets in your game. If you own an Advanced License (and therefore have access to the stripping option), the empty scene with just the main camera can be reduced to about 6 MB in the AppStore (zipped and DRM attached) and you will have about 14 MB available for compressed assets.

Why did my app increase in size after being released to the AppStore?

When they publish your app, Apple first encrypt the binary file and then compresses it via zip. Most often Apple's DRM increases the binary size by about 4 MB or so. As a general rule, you should expect the final size to be approximately equal to the size of the zip-compressed archive of all files (except the executable) plus the size of the uncompressed executable file.

Page last updated: 2011-11-07