For more information, discussion and feedback of releasing EaaS projects, please refer to the CRYENGINE Community thread.
This document applies to CRYENGINE version 3.8.6 and newer,

Overview

Chapters:

Related Pages:

- Getting Started with Engine Code
- Getting Started with WAF

Sometimes, you want to ship the most optimized and most secure binaries and assets for a project to your end-users.

Consider also, if you don't particularly care about cheating/security (such as in single-player games, or proof-of-concept projects), these things may unnecessarily complicate your project.

This document assumes you already have a working game in "Profile" configuration, complete with all the assets and debugging of the code already done.

This document details shipping an EaaS based project to end-users, it is not necessary for internal development, and it also doesn't apply to "full" Engine code licensees packages, FreeSDK packages or other non-EaaS packages.
If you don't care about maximum performance or security, you needn't follow these steps at all. This entire document is mostly relevant for optimized use cases for projects nearing their shipping dates.

One-time setup

Setting up cryptography keys

Open a command prompt, and make `<root>/Tools/PakEncrypt` the current directory.

Now run: `KeyGen.exe`

This should create a `key.dat` and `key.h` file.

Never give out your key.dat file to anyone you don't trust.
If you lose the key.dat file, any PAK files that have been encrypted with it, can no longer be decrypted.
Crytek cannot help you to recover a lost key.dat file.

In your favorite C++ IDE open the generated `key.h` file, and also `<root>/Code/GameSDK/GameDll/GameStartup.cpp` (or the same file in your game DLL).

Now find this line in the GameStartup file:

```
#define USE_RSA_KEY
```

Make sure that USE_RSA_KEY is set to 1.

Replace the key in `GameStartup.cpp` with the key from the generated `key.h` file.

It's not required to provide the RSA key for profile or debug configurations.
Not providing a key can reduce the initial start-up time by ~1 second, but binaries from that configuration can no longer open encrypted PAK files.

Setting up CVar whitelist

In the file `<root>/Code/GameSDK/GameDll/GameStartup.cpp` there is a function:

```
CCVarsWhiteList::IsWhiteListed
```

Here, you can add/remove CVars that may be modified by the end-user (or via system.cfg etc).

Any CVar that is not listed will be fixed in it's default value when in Release mode.

To prevent trivial cheats, it may help to keep this list as small as possible.

However, think of the target machines of end-users and don't freeze quality CVars at too high quality levels.

Creating a Release build

Compilation

You should compile your code in the `[GameSDK] Release` (and `[GameSDK Server] Release` if you use dedicated servers in your project) configuration for your target platforms.

You will get several DLL files in a `<platform>_release` folder if this succeeds, that reflect the optimized and better secured versions of their non-release counterparts.

Don't mix release with non-release DLLs (ie, don't rename DLLs), because they are NOT compatible and the end result will be unstable, if it runs at all.
Combine the release DLLs with the release EXE only, and combine the non-release DLLs with the non-release EXE only.
Release DLLs cannot be used/loaded from the editor (and therefore, there is no Editor.exe in the release folders)

Packed assets

We will not go into detail in this document on how to pack your assets, but we assume you have all the assets required by the game packaged into appropriate PAK files.

After a release build, all assets must be inside a signed or encrypted PAK files.
The first test to see if your PAK files are alright would be to run the non-release game after deleting all non-PAK files.

See also Compiling Assets for Multiple Platforms and the RCJob_Build_SDK.xml that has been shipped to you.

Collecting files into staging

Copy ONLY the following files to a new folder that will contain all the files for the final version of your project (ie, the distribution)

Files and folders marked in red should NOT be copied, but are provided in the list for completeness sake.

For the sake of example, this will only show binaries for 64-bit Windows support. For 32-bit Windows and Linux support, substitute the appropriate platform instead of "win_x64" in the folder names below, e.g. win_x86 for 32-bit Windows or linux_x64_clang for Linux using Clang as the compiler.
On Linux, note that some files will be named slightly differently (e.g. libXXX.so instead of XXX.dll).

Folder	Files	Notes
bin\win_x64_release	CryXXX.dll	Compiled from your game code
	GameXXX.exe	Shipped by Crytek in EaaS package **Note**: You can rename this exe if you want
	*.dll	Dependencies, shipped by Crytek Windows releases If you target your project at high-end computers, you may want to consider removing the bin\win_x86_release folder entirely. This requires that the end-user has a 64-bit machine with a 64-bit OS installed, which is common nowadays. By skipping the bin\win_x86_release folder, you can reduce the amount of testing required on your project.
bin\win_x64 and bin\win_x64_dedicated		**NEVER copy these folders!** These files are debug/profile builds and not suitable for release. Additionally, the Editor is included in bin\win_x64, which you cannot redistribute Editor to end-users according to your EULA. Also, Steam DRM protection is present in the Editor that would make it unusable to anyone without an EaaS license.

bin\win_x64_dedicated_release	CryXX.dll	Compiled from your game code You can skip the _Dedicated folders entirely if you don't want/need end-users to host servers. In addition, if no multiplayer is required, you can also skip the entire _Dedicated folders Linux support This folder is only relevant for Windows; there are currently no dedicated server binaries available on Linux.
	GameSDK_Server.exe msvcp110.dll msvcr110.dll	Shipped by Crytek in EaaS package
Engine	Engine.pak Shaders.pak ShadersBin.pak ShaderCache.pak ShaderCacheStartup.pak	Shipped by Crytek in EaaS package
GameSDK (or `sys_game_folder` CVar value)	game.cfg PAK files, e.g.: Animations.pak GameData.pak GeomCaches.pak Objects*.pak Scripts.pak Sounds.pak Textures*.pak Videos.pak	Your packaged assets - the exact list of PAK files will depend on how you organize your assets; the listed files represent the structure in the EaaS package by Crytek **Note**: Other than inside the levels folder (see below), all assets should be inside a PAK file.
GameSDK/Levels (or relative to `sys_game_folder`)	Each level folder (e.g. Singleplayer/Woodland) may have: filelist.xml <Levelname>.dds <Levelname>.xml level.cfg level.pak Occluder.ocm <LoadingScreen>.dds (if you have defined one)	Your exported levels **Note**: Don't copy the .cry file, the terraintexture.pak, the Layers or LevelData sub-folders The files in here, with the exception of level.pak, must be inside some (signed) PAK file in order to load the levels with the Release build. However, only having the data in PAK files will break some editor functionality. For this reason, we ship some of these files both inside and outside PAK files (with identical content).
Localization	<Language>.pak <Language>_xml.pak	Your supported languages for which your project is localized
Code Editor Tools User LogBackups TestResults	<all files and sub-folders>	These folders contain no end-user usable files, save some space by not shipping them The user folder should only have user-specific settings and will be regenerated automatically by the end-user If you have settings that you expect all users to have, consider moving them to system.cfg instead of user.cfg
<root>	system.cfg	Make sure the `sys_game_folder` CVar is set in this file
	game.log editor.log server.log error.dmp	Do not ship these files: it will make it harder to identify if a log-file was user generated or not

Additional files specific to your project may have to be added as well, in which case it's your responsibility to ensure they are added in the correct location and are loadable.
Consider putting those files inside a custom PAK file, if you want to leverage the existing signing/encryption features.

Carefully consider legal obligations on re-distribution that may apply on software and assets as well, make sure to read license agreements etc.

Some additional space can be saved if you can ensure that the Visual C++ 2012 runtime is installed on the end-users machine during project deployment / installation.
In that case, you can remove the msvcr110.dll and msvcp110.dll in each bin folder, since they will be present on the system.
Note for users that are not using Visual C++ 2012 compiler, make sure that runtimes (if any) are either shipped or are installed in some other way.
Even if you don't use the Visual C++ 2012 compiler, we used that one to compile the _Release.exe, so the runtimes for 2012 still need to be present or installed (which causes duplicate runtime requirement)

We recommend against mixing runtimes from different Visual C++ editions, since it can (theoretically) cause hard to diagnose bugs.
If possible, consider building your releases with the same compiler as Crytek used for compilation to minimize chances of problems.

Signing / Encryption of assets

Once you have your staging folder set-up, you need to either sign OR encrypt your asset PAK files.

Depending on your project, you should pick one of those:

- Signed assets can be loaded by your project from the PAK files, and can also be opened with other tools (as a ZIP file)
  However, the PAK file cannot be modified after it is signed (or the engine will refuse to load the files)
- Encrypted assets can be loaded by your project from the PAK files, but cannot be opened with other tools.
  Assets inside the PAK cannot easily be accessed because they are encrypted (but can be decrypted at run-time by the engine)

Since the engine can decrypt the encrypted assets at run-time off-line, this is not a 100% secure way to protect your assets if the project is intended to be run off-line.
Consider though, that if at some point an asset is displayed on screen, an attacker can recover the asset data using (for example) a graphics profiler such as RenderDoc or PIX regardless of any encryption, or read the decrypted data from main memory with a debugger.
A signed or encrypted PAK file cannot be renamed after signing/encryption, because the filename (excluding folder names) are part of the signature.

Once you have decided what level of security you want:

- Pick an output folder for this project build.
- Open a command prompt, and make `<root>/Tools/PakEncrypt` the current directory (the same folder that has the `key.dat` from the one-time setup).
- Now run: `dist/ParseBuild/ParseBuild.exe <verb> "<full path to your staging folder>" "<full path to an output folder>"`
  - verb can be `sign` or `encrypt`, depending on the method you picked

Now the files from the staging folder will be copied from the staging folder to the output folder.

You can now delete the staging folder to recover some disk space.

As an optimization, you can also copy only PAK files into the staging folder (keeping the correct folder structure), and copy all non-PAK files directly to the output folder (with the same structure), which saves a copy operation for those files.
However, that's outside the scope of this document, and since a significant relative amount of data volume is in the PAK files anyway, it might not be worth the additional complexity to your build systems.

Test and redistribute

Now you can take the contents of the output folder and test or redistribute it.

You should in general make sure your entire project works when you directly run any of the .exe that is in the output folder, without adding any additional files.

Testing this ensures that all your code and assets are present and functional for end-users.

Try to run the project on a variety of hardware configurations to find out quality settings for your minimum advertised hardware specifications.

If you wish to be able to debug crash-dump files that are generated by your project, make sure to keep a copy of all the .pdb files that are matching the binaries that you redistribute (for each version that may be live).
That way, you can load error.dmp files from end-users and have (limited) information that might point you to the cause of the crash, in addition to the (limited) log files.

These .pdb files needn't be distributed as long as you can find the matching ones when loading the end-user's dump file.
Redistributing via a specific publishing platform (ie, Steam) is outside the scope of this document. You should probably read the documentation provided by the publishing platform to find out how it works.
In general, as long as the end-users folder structure matches the one that is described here, there shouldn't be any issues.

The same comment applies to installers of various kinds, as long as the directory structure is maintained and all the files mentioned above are present, the project should be runnable.
It's important to also include your redistribution method in your testing; if you publish on Steam you should test the game downloaded via Steam, if you publish with an installer, you should also run the installer before testing.
This helps finding problems due to missing files, or files installed to a wrong folder.

Troubleshooting

Sometimes it's hard to debug issues with Release builds, because code is optimized and assets may not be easily accessible.

Here are some pointers to isolate problems:

My game doesn't run in Release mode

Make sure that:

- You have your PAK files signed or encrypted.
- You have the correct key set in the code.
- The profile/debug version of the game does run with the same assets (and the same key).

My CVars don't work in Release mode

Make sure that:

- The CVar in question is white-listed in your code.

My assets cannot be found in Release mode

Make sure that:

- The asset exists in a PAK file, and the PAK file is in the list of PAK files to load in the code.
- The PAK file containing the asset is signed or encrypted with the correct key.

I get a lot of warnings about "Non binary XML found"

This is not a fatal error, however you can convert your XML files to binary XML using RC, see also Binary XML conversion
The conversion is recommended for an end-user distribution

I cannot connect to a dedicated server

Please note that you should connect release clients with release servers. (And also, you should connect profile/debug clients with profile/debug servers)

I have some other problem

The default value of log_severity in release builds are quite low, so to debug an issue it might make sense to set it to 3 or 4 to get more information and messages.
Note that, unless you whitelist this CVar, setting it in system.cfg will not actually work.