Overview

The Game Code package supply the GameDLL C++ source code along with the required header files to access most of the CRYENGINE systems. It is stripped down for users who don't have access to the full Engine Code.

Starting with 3.6.3, users of the Game Code package now also get additional access to the CryAction and CryInput projects, including all the source files required to build CryAction.dll and CryInput.dll for Windows platforms.

CRYENGINE licensees who obtained a full Engine Code License should only use the Engine Code package which already includes all the files supplied in the Game Code package.

The GameDLL provides developers a project where they can implement a game class based on the IGame interface. It's also possible to implement new Entity class and extensions to game systems (i.e. vehicle movements, firemode or AI characters).

For the Steam release, the code will be provided inside a .zip file, placed in the root folder of your installation. The reason the files are placed in a .zip is so that any changes you make are not overwritten when steam updates your installation.
After an update, you can either merge the contents of the new .zip file into your existing code (using source control or other tools of your preference), or just replace your code with the new code if you have not made any changes.

Prerequisites

Required Software

Please refer to Visual Studio supported versions for information on which version of Visual Studio and Windows SDK are supported.

Pre-CRYENGINE 3.6.3

Before 3.6.3: When using the Game Code package, you don't need to install any additional 3rd party SDKs, all the required SDKs for building the GameDLL are included in the source code package.

CRYENGINE 3.6.3 -> 3.6.15

Starting with 3.6.3: In order to build the CryInput.dll, you need to obtain the DirectX SDK from Microsoft here (it's free). After installation, you must copy the contents of the installed SDK (the Include and Lib folders) into <CRYENGINE root>\Code\SDKs\DXSDK

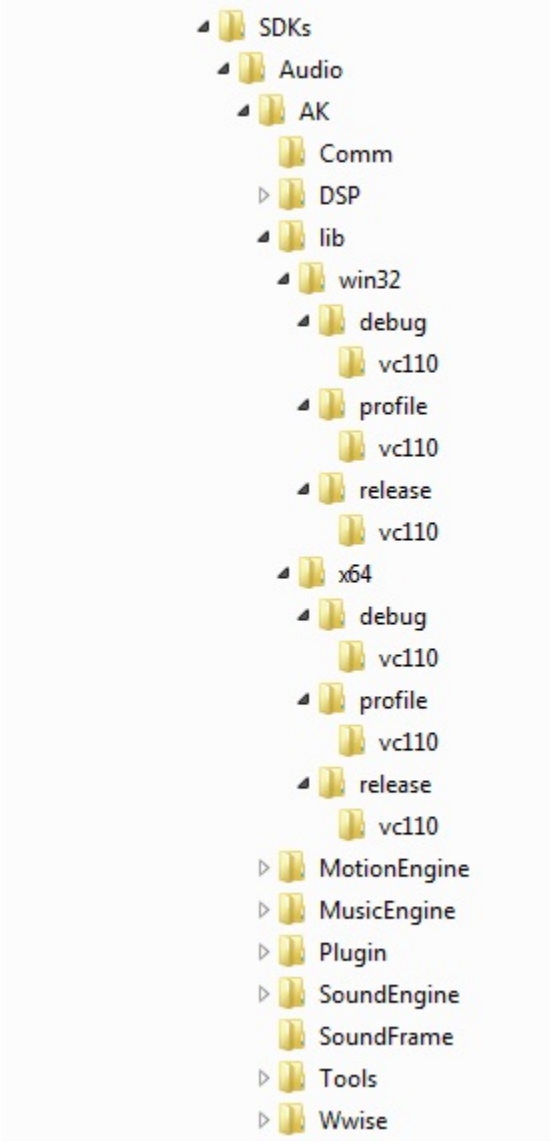
Alternatively, before building the Game code solution, disable the CryInput project from building by unchecking the check-box in the solution Configuration Manager for all combinations of Configuration and Platform.

CRYENGINE 3.6.9 -> Current

Starting with 3.6.9: In order to build the CryAudioImplWwise.dll, you need to obtain the Wwise SDK from Audiokinetic here. After installation, you must copy the folder <Wwise Installation root>\SDK\include\AK to <CRYENGINE root>\Code\SDKs\Audio\AK. Then create the <CRYENGINE root>\Code\SDKs\Audio\AK\lib folder and copy the contents of the following folders:

From	To
<Wwise Installation root>\SDK\x64_vc110\Debug\lib	<CRYENGINE root>\Code\SDKs\Audio\AK\lib\x64\debug\vc110
<Wwise Installation root>\SDK\x64_vc110\Profile\lib	<CRYENGINE root>\Code\SDKs\Audio\AK\lib\x64\profile\vc110
<Wwise Installation root>\SDK\x64_vc110\Release\lib	<CRYENGINE root>\Code\SDKs\Audio\AK\lib\x64\release\vc110
<Wwise Installation root>\SDK\Win32_vc110\Debug\lib	<CRYENGINE root>\Code\SDKs\Audio\AK\lib\win32\debug\vc110
<Wwise Installation root>\SDK\Win32_vc110\Profile\lib	<CRYENGINE root>\Code\SDKs\Audio\AK\lib\win32\profile\vc110
<Wwise Installation root>\SDK\Win32_vc110\Release\lib	<CRYENGINE root>\Code\SDKs\Audio\AK\lib\win32\release\vc110

The final folder structure in <CRYENGINE root>\Code\SDKs\Audio\ should look like this:



Using STLport

STLport is an open source implementation of the Standard Template Library (STL) different from the default implementation shipping with Visual Studio.

Usage of STLport is no longer recommended since the improvements in the Visual Studio shipped STLs.

The STLport source code is included in the CRYENGINE SDK. However, Visual Studio needs to be set up to make use of STLport instead of the default STL implementation using the following steps:

- Go to **Tools/Options** in the main menu and select **VC++ Directories** under **Projects and Solutions**.
- Select Platform Win32 and add the path <CRY_SDK_ROOT>\Code\SDKs\STLPORT\stlport at the top of the **Include files** and **Library files** list. It is essential that the STLport path is above the default Visual Studio include and lib paths in the list.
- Repeat the previous step for the x64 and other platforms, as required.

Solution Files

This information applies to CRYENGINE before 3.7.0
In CRYENGINE versions since 3.7.0, WAF is used as the build system.
Please read WAF Build System on information how to generate Visual Studio solution files.

The following table describes the solution file supplied with the CryENGINE SDK.

Solution file	Description
`Code\Solutions\CryEngine.sln`	Solution delivered to full Engine Code licensees.
`Code\Solutions\CryEngine_GameCodeOnly.sln`	Solution delivered to other users. *Starting with 3.6.3:* This solution also includes code for CryAction and CryInput

The CryEngine.sln solution is only supplied with the full engine source code. The content of the CryEngine_GameCodeOnly.sln solution is included in CryEngine.sln.

The Game Code solution can always be compiled in either Debug or Profile configuration. Compiling the Release configuration requires the access to full engine source code.

Source files

Directory	Description
`Code\CryEngine\CryCommon`	Headers for all the interfaces of CRYENGINE.
`Code\CryEngine\CryAction`	Include headers for the Game Framework which include the implementation of Flowgraph, Vehicle, etc. *Starting with 3.6.3:* Includes all source code for CryAction.dll, so users can build it from scratch.
`Code\CryEngine\CryInput`	*Starting with 3.6.3:* Includes all header and source files for CryInput on Windows.
`Code\CryEngine\CrySoundSystem`	*Starting with 3.6.9:* Includes all header and source files for building the CryAudioImplWwise module.
`Code\GameSDK\GameDll`	Source files for the reference game shipped with the SDK.

The header files from CryCommon and CryAction are required to recompile the GameDLL. More information on CryAudioImplWwise and writing your own Audio Implementations can be found here.

Where to start

Initialization

The code handling the initialization of a CRYENGINE game is contained inside `Code\GameSDK\GameDll\GameStartup.cpp`. The function *CGameStartup::Reset()* is handling the allocation and initialization of the IGame interface.

For adding any game specific initialization, it is recommended to look into *CGame::Init()*. This function is called once when the game is loaded. The function *CGame::Shutdown()* will be called when the game is being shutdown.

Frame Update

Any function that needs to be updated for every frame should be added to one of the two update functions.

```
int CGame::Update(bool haveFocus, unsigned int updateFlags)
```

```
void CGame::OnPostUpdate(float fDeltaTime)
```

Debugging

After making and building code changes in the GameDLL or in another project, you will probably want to be able to debug the new code (or any other code).
There are two ways to debug from a game code project

- Edit the debugging settings of the CryGameSDK project.
  - Right-click the project in the solution explorer, and select properties
  - Under configuration properties category, go to Debugging
  - Update "Command" to point to GameSDK.exe (for launcher) or Editor.exe (for Sandbox) on your local computer. Pick the .exe from Bin64 if Platform is set to x64, or from Bin32 if Platform is set to Win32 (top right drop-down)
  - Update "Working Directory" to point to the root folder of your CRYENGINE installation. (This is the the folder containing the Bin32 and Bin64 folders)
  - Set the CryGameSDK project as start-up project. Right click the project in the solution explorer, and pick "Set as StartUp Project"
  - You can now debug the game code in either Sandbox or the launcher (depending on which you picked two steps up) by using Debug -> Start debugging (or press the default shortcut, F5)

- Attach the debugger to an existing process.
  - Start either the launcher or the sandbox after building your code
  - In visual studio, go to Debug -> Attach to Process
  - Find either GameSDK.exe (for launcher) or Editor.exe (for Sandbox) in the list of processes. Then attach to it.
  - You can now debug the game code in the running process

If you have obtained your installation of CRYENGINE using Steam, you can only use the second method for debugging game code in the Editor. You can still use the first method to debug code in the Launcher.

Common Questions For Beginners

Q: I'm getting an error about "Unable to start program... /CryCommon" when building the GameCode solution, why?

A: Users without full Engine Code will only be able to build "CryGame", so make sure that's what you're building.

Q: I finished building CryGame but at the end it gave me an error about "Unable to start program... /CryGame.dll", why?

A: If you have set CryGame as your "Startup Project" then upon completion of the build, it will attempt to 'run' it, which can't happen.