UHC Plugin System

Since its version 1.5, the UnHardcode Patch allows modders to create custom modules, called “plugins”, which are custom DLLs that follow a specific coding specification and can be used to apply additional custom patches and implement custom game syscalls and cheat codes, using native UHC Patch features.

How do UHC Plugins work

When loading a plugin, the Patch expects it to have an `UHCPluginMain` function, which is responsible for the actual plugin initialization. In case this function is found, the Patch calls it and passes to it a pointer to an `UHCPluginInfo` structure, which contains pointers to native functions, which can be used to register custom game syscalls and cheat codes.

This function, after being called, should just return 0.

Creating your own UHC Plugin

In order to develop a UHC Plugin, just follow closely the steps below:

1. Create a DLL project using your favorite C/C++ IDE (we recommend Visual Studio)
2. Include the “UHCPlugin.h” header in your project
3. If needed, change the source code of the header, so your IDE or compiler can recognize it properly. Although, if you do any changes, make sure to keep the types as consistent as possible
4. Create an `UHCPluginMain` function that uses the stdcall calling convention receives a pointer to an `UHCPluginInfo` structure as argument and returns 0, and define it as exportable. Its declaration should be similar to this:

   ```
   extern "C" _declspec(dllexport)
   int __stdcall UHCPluginMain(UHCPluginInfo* pluginInfo)
   ```

5. Program this function as you will
6. Configure your IDE or compiler, so, after compiling the code, it generates a DLL with a “.upl” extension, instead of the default “.dll” extension
7. Compile the project
8. In case your IDE or compiler didn’t allow you to set a custom extension, make sure to rename the extension of the compiled DLL manually to “.upl”, otherwise **the patch will not load the plugin.**
9. Place the compiled plugin in the root folder of your AoE3 installation, that is, in the same directory of the main executable and of the DLL of the Patch.

Attention: Make sure to explicit the calling convention used by the UHCPluginMain function, in order to ensure proper functionality.

To implement custom syscalls and cheat codes using plugins, refer to the following sections.

Custom Syscalls

Syscalls are hardcoded native functions which are used in AIs, RMs, Trigger Scripts and Interface files and other contexts of the game to perform specific actions. The UHC Patch allows the implementation of custom Syscalls through the Plugin System.

Implementing Custom Syscalls

To implement custom syscalls using the UHC Plugin system, follow the steps below:

1. Create a valid UHC Plugin using your favorite C/C++ IDE
2. Include the “syscalls.h” and “syscalls.cpp” source files in your project
3. Code a C/C++ function for your syscall. This function doesn’t need to be exported and can only receive as arguments variables of the types integer, float, boolean, string (as defined in syscalls.h) and vector (as defined in syscalls.h. Its return value should either be void or belong to one of the aforementioned types. This function can also call the existing AoE3 syscalls, as long as they belong to a compatible scope
4. In your plugin’s `UHCPluginMain` function, call the `RegisterSyscall` function (the `UHCPluginInfo` structure pointer contains a pointer for that function), while setting the parameters passed to that function accordingly.
5. If you want your custom syscall to have default values for its parameters, call the `SyscallSetParam` function to define default values for the parameters that your syscall uses
6. Add a `customSyscalls` entry to the UHC configuration file used by your mod, with no values, so the Patch can load the implemented custom syscalls

For an example of a UHC Plugin that implements custom syscalls, refer to the “UHCPluginDemo” Visual Studio project, which can be found at the “Examples” folder.

Custom Cheat Codes

Every AoE3 cheat code is stored, internally, as a structure that contains a pointer to an encrypted Unicode string that refers to the cheat name, a flag that indicates whether the cheat is enabled or not and a pointer to a function that is executed when the cheat is detected.

To implement custom cheat codes through the UHC Plugin system, just follow the setps below:

1. Create a valid UHC Plugin using your favorite C/C++ IDE
2. Include the “syscalls.h” and “syscalls.cpp” source files in your project, if you want to use any of the game’s syscalls for your cheat effect
3. Code a C/C++ function for your cheat. This function doesn’t need to be exported, must use the CHEATCALL calling convention (which wraps the __thiscall calling convention in a way that is friendly for most C/C++ compilers) and will receive a pointer to the current player Data as an argument when called by the game. An example of a possible declaration for such function can be seen below:

   ```
   void CHEATCALL testCheat(void* playerData)
   ```

   The cheat function can call any of the syscalls that are callable during a game, as long as the parameters are set accordingly. It can also call the `CheatAddResource` and `CheatSpawnUnit` functions, which are used in some of the default cheats.

4. In your plugin’s `UHCPluginMain` function, call the `RegisterCheat` function (the `UHCPluginInfo` structure pointer contains a pointer for that function), while setting the parameters passed to that function accordingly. The Unicode string that is passed as an argument should be the text that the player must type in order to activate the cheat, which will be encrypted by the Patch before being stored in the internal game data.
5. Add a `customCheats` entry to the UHC configuration file used by your mod, with no values, so the Patch can load the implemented custom syscalls

Attention: Make sure to explicit the calling convention used by the cheat function, in order to ensure proper functionality.

For an example of UHC Plugins that implements custom cheat codes, refer to the “UHCPluginDemo” Visual Studio project at “Examples” folder.