Subject: Re: [Request] List of parameters for hud.ini
Posted by Gen_Blacky on Thu, 07 May 2009 14:48:59 GMT
View Forum Message <> Reply to Message

Toggle SpoilerBHS.DLL is a new dll created by Blackhand Studios to go alongside the custom scripts.dll and provide
gameplay fixes, new console commands and other enhancements. Basicly, all the stuff we are doing that
requires code to run client-side, sending data over the network or patches to the code.

BHS.DLL adds the following new console commands:
ID displays the name and ID of all players matching a specified string (e.g. ID jon matches jonwil and jonathan and abcjon but not abcjoe). Passing no string means "print all players".
PAMSG sends an admin message to a particular player
PPAGE sends a page to a particular player
SNDP plays a .wav sound (which can be from the always.dat of whichever mod is involved or one stored directly in the renegade data folder) for a specific player
SNDA is like SNDP but plays the sound for everyone
TEAM changes the team of a given player to the specified team. If they are alredy that team, nothing will happen
TEAM2 changes the team of a given player to the specified team. If they are alredy that team, nothing will happen. Unlike TEAM, it doesnt reset their score or cash.
DONATE takes a specified amount of cash from one player and gives it to another player
EXIT exits renegade
VERSION sends a message to the client of the specified player. The client then responds with a message indicating if bhs.dll is installed (and its version), this is then displayed on the console
SNDT is like SNDP but plays a sound for the specified team.
SND3DP and SND3DA like SNDP and SNDA except they are 3D. They take wav files.
TPAGE sends a page to a particular team
MLIMIT changes the mine limit (note that this will NOT be reset at the end of the map)
MUSICP plays a MP3 music file for a particular player
MUSICA plays a MP3 music file for all players
NOMUSICP stops the music for a particular player
NOMUSICA stops the music for all players
SVERSION prints the version of bhs.dll installed.
RADAR prints the current radar mode
MLIST prints the name of the map at a given index in the map list
MLISTC changes the name of the map at a given index in the map list
MAP prints the name of the current map
MOD prints the name of the current mod package
MAPNUM prints the index in the map list of the current map
MLIMITD prints the current mine limit
MINED prints the current mine count for a given team
EJECT ejects the specified player from whatever vehicle they are in (if any)
SONG prints the current song (a call to Set_Background_Music or use of MUSICA will change the song, a call to Stop_Background_Music or use of NOMUSICA will clear the song. The per-player commands have no effect on the current song.)
ICON displays an emoticon above the players head (like the radio commands only with no sound

or text). You can make any w3d file appear. The emoticon is only visible to players on the team of the player whos head it appears over.
WIN ends the game in favor of a particular team by destroying all the buildings of the opposing team
SCREENSHOT changes the format of screenshots output by renegade. 0 = PNG, 1 = TGA. This setting gets saved into the registry. The default (if you have never used the SCREENSHOT command before) is PNG.
See below for details of the new PNG screenshot code
For reference, the setting is saved as ScreenshotFormat under the HKEY_LOCAL_MACHINE\SOFTWARE\Westwood\Renegade key, values are the same as for the console command (0 = png, 1 = tga)
SCREENFMT prints the current screenshot format
LOG changes whether the client chat log is output or not, 0 = disabled, 1 = enabled. This setting gets saved into the registry. The default (if you have never used the LOG command before) is enabled.
For reference, the setting is saved as ClientChatLog under the HKEY_LOCAL_MACHINE\SOFTWARE\Westwood\Renegade key, values are the same as for the console command (0 = disabled, 1 = enabled)
LOGP prints the status of whether the client chat log is being output or not
TMSG will send a team message as though it came from the specified player
GETBW will print the current bandwidth for a player (same thing as what the sbbo command sets)
SETBW will set the current bandwidth for a player (same thing as what the sbbo command sets)
Neither command will send anything to the client.
WOLNAME special command to set the WOL name used for. Dont use this unless you know you need to send the packet. LFDS only.
PINFO new console command to print information about players in the game.
Data is comma delimited and is printed in the following order:
Player ID
Player Name
Player Score
Player Team
Player Ping
Player IP
Player KB/s
Player Rank (position in the player list)
Player Kills
Player Deaths
Player Money
Player Kill/Death Ratio
I wanted to add player time but I havent figured out how to retrieve that information yet
ICON2 displays an emoticon above the players head (like the radio commands only with no sound or text). You can make any w3d file appear. The emoticon is only visible to players on the enemy of the player whos head it appears over.
VLIMIT changes the current vehicle limit
VLIMITD displays the current vehicle limit
CMSG sends a colored text message to the client
CMSGP is like CMSG but private
CMSGT is like CMSG but team

Some notes:
1.The message appears in the same box as messages like "unit ready".
2.Each message sent will appear on its own line (i.e. there is an implicit newline) and has one RGB color associated with it.
3.Note the commas between the red, green and blue color values. They must be commas, not spaces or anything else.
and 4.Instead of 1,2,3 for the color you can use the following special values:
private Private message color
public Public message color
gdi GDI color
nod Nod color
player Color of the player the message is being sent to (CMSGP only)
team Color of the team the message is being sent to (CMSGT only)
Anything else that isnt a proper R,G,B color value will not work.
Note that messages sent this way do not pass through the hosthook or chathook and do not get logged into any logfiles (renlog, bhs_renlog, client chatlog etc)
DISARM disarms all C4 of a player
DISARMP disarms all proximity C4/mines of a player
DISARMB disarms all beacons of a player
RLMON is for the renlogmon feature
RLMONOFF is to turn off the renlogmon feature
PLIMIT is to set the player limit
PLIMITD displays the current player limit
MAXPLIMITD displays the origonal (and maximum) player limit
TIME changes the time remaining
TIMED displays the time remaining
TIMEL changes the time limit
TIMELD displays the time limit
VIEW displays a dialog box containing a w3d file with an animation (simiar to what the encyclopedia displays in SP). This is only for win32 client and only for "I am host". Its intended for testing purposes.
HUD disables or enables the HUD. Its client-side only and is intended for screenshots and movies and such.
MAPCH checks if a player has a given file in their data folder. A message will be printed on the screen if they have it or if they dont have it. No message at all means they have scripts.dll < 3.2

both RLMON and RLMONOFF are for dedicated servers only (i.e. server.dat, LFDS) and not for game2.exe

EXIT, LOGP, LOG, SCREENSHOT and SCREENFMT run client-side only.
SVERSION runs both client-side and server-side.
everything else is server-side only.
Note that as of 2.7.x, the ID console command is server-side only.
The following commands require bhs.dll on the client, the rest do not:
SNDP
SNDA
SNDT
SND3DP

SND3DA
SND3DT
MUSICA
MUSICP
NOMUSICA
NOMUSICP
ICON
VLIMIT
CMSG
CMSGP
CMSGT
TIME
TIMED
TIMEL
TIMELD
MAPCH

Also, BHS.DLL fixes the following script commands to work in multiplayer:
Set_Animation
Set_Animation_Frame
Enable_Stealth
Create_Explosion
Create_Explosion_At_Bone
Set_Fog_Enable
Set_Fog_Range
Set_War_Blitz
Fade_Background_Music
Set_Background_Music
Stop_Background_Music
Create_Sound
Create_2D_Sound
Create_2D_WAV_Sound
Create_3D_WAV_Sound_At_Bone
Create_3D_Sound_At_Bone
Play_Building_Announcement
Clear_Weapons (only for vehicles, infantry dont need it)
Enable_Vehicle_Transitions
Set_Player_Type
Set_Screen_Fade_Color
Set_Screen_Fade_Opacity
Set_Display_Color
Display_Text
Display_Float
Display_Int
Select_Weapon
Shake_Camera
Set_Obj_Radar_Blip_Shape
Set_Obj_Radar_Blip_Color

Display_Health_Bar
Disable_All_Collisions
Disable_Physical_Collisions
Enable_Collisions
Control_Enable

BHS.DLL also adds versions of the following script commands (named with _Player at the end and called directly without the Commands-> in front)
which take a GameObject and activate for the player represented by that object. Should that object not represent a player, nothing happens,.
Set_Fog_Enable
Set_Fog_Range
Enable_Stealth
Fade_Background_Music
Set_Background_Music
Stop_Background_Music
Enable_Radar
Display_GDI_Player_Terminal
Display_NOD_Player_Terminal
Set_Screen_Fade_Color
Set_Screen_Fade_Opacity
Force_Camera_Look
Enable_HUD
Create_Sound
Create_2D_Sound
Create_2D_WAV_Sound
Create_3D_WAV_Sound_At_Bone
Create_3D_Sound_At_Bone
Set_Display_Color
Display_Text
Display_Int
Display_Float
Set_Obj_Radar_Blip_Shape
Set_Obj_Radar_Blip_Color

All of the commands except Create_Explosion and Create_Explsion_At_Bone require the client to have BHS.DLL (and scripts.dll) installed.
In some cases, not having it will render the map useless (e.g. they miss out on being able to access PTs because of the Display_Player_Terminal
commands or they dont get their radar disabled by Enable_Radar and gain an advantage) but in others (e.g. if Create_Sound is used to play sounds), it doesnt matter.
As mentioned below, use JFW_BHS_DLL if you want BHS.DLL to be required for the map.

Note that Set_Display_Color, Display_Text, Display_Float, Display_Int and the per-player versions of same are a little flakey and may not work right.

Also, BHS.DLL contains working Poke in multiplayer. This means that a script can respond to the Poked event and machines

other than the host can walk up to the object and use the action key on it. Like the script command fixes, this only works
if the client has BHS.DLL installed.
Also, BHS.DLL contains fixes (that require bhs.dll on the client) to make the Set_Model script command function correctly for vehicles and for infantry.
And it contains a fix so that giving a weapon powerup to a vehicle will function correctly. (which also needs bhs.dll on the client)

And, it has a fix that makes the harvester harvesting arms animation work correctly (this requires bhs.dll on the client for the animation fix)
Note that if you build a map that uses features such as Poke or Stealth or other "client is required" items
(i.e. if the client doesnt have bhs.dll installed, the map is unusable or not having it gives an advantage like with Enable_Stealth),
put the script JFW_BHS_DLL on an object in the map somewhere (something like a special Daves Arrow is good since its created on map startup).
This script prints a message to the console which regulators such as BrenBot can use to kick players without BHS.DLL installed.

BHS.DLL also features working radio command icons in multiplayer for all clients who have BHS.DLL installed.

BHS.DLL contains code to make renegade read different files instead of the *.dep/always.dep files.
This means that map loading is much faster.

BHS.DLL contains a new hook system that will be triggerd on the server when an object is created but before any observers on that object have run
To install an object create hook, you create an ObjectCreateHookStruct and fill it in with the address of the hook function plus a user-specified data pointer
(can point to whatever you like, it gets passed back to the hook function).
Then you pass the ObjectCreateHookStruct to AddObjectCreateHook.
To remove the hook, you pass the return value from AddObjectCreateHook to RemoveObjectCreateHook.
The hook will then trigger everytime an object is created.
Although there are a few instances where the Object Create Hook might not trip (e.g. certain spawner objects). This is because those objects dont
call through to ScriptableGameObj::Start_Observers (the place where the scripts get started up and the place I am hooking)

The hook function looks like this:
void ObjectCreateHook(void *data, GameObject *obj)
the value of data is what you passed in when you registered the hook.
the value of obj is the value for the newly created object.
Objects that are on the map at startup do not go through these hooks.
Look at JFW_Object_Create_Hook_Base and related scripts in jfwhook.h/jfwhook.cpp to see an example of how to use this.
This wont catch objects already on the level at startup (which can include players under some

circumstances e.g. "host = client")
So, you have to do whatever it is you want to do on those objects manually.

BHS.DLL also contains code that replaces the screenshot code in renegade with new code.
This new code outputs screenshots in PNG format instead of TGA format.
It has been known to cause lag when it is being used, this is a side-effect of the time libpng takes to compress the PNG file and there isnt anything I can do about it.
A PNG file is a compressed image file similar to a JPEG file.
Most image editors that can read and write JPEG files can read and write PNG files.
Web browsers can also display PNG just as easily as they can display JPEG.
PNG was chosen because it takes less time to compress than JPEG.
Plus, the code for compressing PNG files is smaller than that for JPEG
files (which means BHS.DLL is smaller).
Note that the time taken to read the screen out of your graphics card RAM is also a factor in the sceeenshot lag.

BHS.DLL also contains new keyboard hook code.
On the client (including the server if it is also a client), at startup the keyboard hook code reads in a config file called keys.cfg
Each line in the file looks like this
Jetpack=Y_Key
The part before the = is the logical key name and is what is used by a scropt to refer to the key.
The part after the = is the physical key name and refers to the actual keyboard key tat you press to trigger the particular logical key.
The data is read in from keys.cfg and stored in a structure matching logical keys to physical keys.
Everytime renegade polls the keyboard, the new code iterates through this structure and retrieves the current state of the key.
Then, it goes through some code to make sure that the key hook is only triggered once for every time you press and release the key.
After this, it goes into another hook that checks to see which keys have been pressed (and therefore the server needs to know about the keypress).
It has code to make sure it doesnt trigger when the main menu, dialogs, message box (f2/f3) and console input box are active.
(i.e. if you press the key when those are active, the hook wont trigger)
To install a keyboard hook, you create a KeyHookStruct. This contains the address of the hook, the name of the logical key you are interested in,
the player ID of the player you are interested in plus a user specified data pointer.
(can point to whatever you like, it gets passed back to the hook function).
Then you call AddKeyHook to add the keyboard hook, passing in the KeyHookStruct.
To remove it, you call RemoveKeyHook and pass the return value from AddKeyHook.
The hook function will then trigger every time the specified player presses whatever key they have assigned to that particular logical key (in keys.cfg)
The hook function looks like this:
void KeyHook(void *data)
the value of data is what you passed in when you registered the hook.
Look at JFW_Key_Hook_Base and related scripts in jfwhook.h/jfwhook.cpp to see an example of how to use this.

see keys.txt for details of the valid key names for the keys.cfg config file.
see keycfg.txt for details of the GUI key config application.

bhs.dll also contains code for custom scopes.
Basicly, each infantry unit in your mod can have a different custom scope
Scopes are per-unit (as opposed to per-weapon) because of engine limitations.
There is no real limit on the number of scopes you can have.

Firstly, you create a scope texture for each scope that is square with the width and height a power of 2 (like any other renegade texture). This texture will be "stretched" to fill the screen. Use alpha blending like on the default renegade sniper scope to create the "hole" that you can see through.

Then, you create a scopes.cfg file either in your data folder or in one of the mix files for your mod.
This has lines like the following:
scope.tga=Sniper=0.61154997=11.200000
The first part is the texture name (use .tga on the end even if it is a dds file)
The second part is the name of the camera profile to use
The third part is the minumum zoom level to use (how far to zoom out)
The last part is the maximum zoom level to use (how far to zoom in)
The camera settings shown above provide an acceptable camera profile (similar to how the default sniper scope works)

At startup, scopes.cfg is read with each scope (i.e. each line) being given a number starting from 0.
Then, in your mod you put the script JFW_Scope on every player-buyable infantry unit in the game (i.e. anything the player could become) passing either the scope number of -1 (for no scope).
If you wish to change scopes at runtime, you can use something like JFW_Attach_Script_Custom to attach a new copy of JFW_Scope to change the scope.
JFW_Scope sends a message to the client that initalizes the specified scope.
You can also use MDB_Weapon_Scope (see the readme file for details) to implement per-weapon scope logic.

When the "scope" logical key (set in keys.cfg and used via my keyboard hook code), if the scope is not -1 and the player is not in a vehicle, the scope is switched on. Pressing it again switches it off.
Whilst in scope mode, you can use the "zoom" keys (same as with the normal sniper scope) to zoom in and out.
Also, when the scope is enabled, the Next Weapon, Previous Weapon, First Person Toggle and Action keys are disabled. The weapon switch keys are disabled because the mouse wheel is used for both zoom and for switch weapon. The number keys at the top still work to switch weapons.
The First Person Toggle and Action (i.e. PT, ladder, vehicle entry etc) keys are disabled because they change the camera settings which breaks the scope camera logic. (although if you need to use those keys, just switch out of scope mode)
Some notes:
If you want the normal scope on a unit, put a custom scope of -1 for it. Also, make sure that there is no way for that unit to get a custom scope as having both on the unit will break things. (or just use MDB_Weapon_Scope)

If you put MDB_Weapon_Scope on a unit, dont put JFW_Scope on it and vice versa. If you are using MDB_Weapon_Scope, put that on all units.
All infantry needs either JFW_Scope or MDB_Weapon_Scope on it otherwise bad things will happen.

If you are doing scripting, you can call Set_Scope to change the scope assigned to a player.

There is new logic to change the main HUD texture. Basicly, you put the script JFW_HUD on every infantry unit, passing in the name of a HUD texture.
A HUD texture is a file that is the same layout as HUD_MAIN.TGA. If you want a unit to use HUD_MAIN.TGA, you can pass that, otherwise pass the name
of whatever custom texture you have created for your mod. Every infantry unit must have JFW_HUD on it with some texture otherwise bad things will happen.
When passing the texture, put .tga on the end even if it is a .dds file.

If you are doing scripting, you can call Set_HUD_Texture to change the HUD texture assigned to a player.

There is a new logfile output by bhs.dll with the same name as the regular renlog but with bhs_ at the front.
On windows (client and WFDS) and linux, this will contain file will contain all f2/f3 console messages
In addition, on linux this will contain all team change messages
As of 2.9, the months for this log match renlog (with january starting at 1) on windows.
On linux, the months match renlog too (on linux, renlog months start at 0)

On the client (dont know if its "host is a client also" or not), there is a log called like the renlog but called client_ instead of renlog_.
This will contain the following:
all uses of MESASGE on the host
all uses of PPAGE on the host for this player
all uses of TPAGE on the host for this team
all uses of TMSG on the host for this team
all f2 chat messages
all f3 chat messages for this team
There will be an indication if its for everyone, team or private (note that because of how it works, a TPAGE command will be marked "private" and not "team", messages sent by the TMSG command will show up as team messages)
This also records messages like "host: xxx changed teams" and "host: xxx committed suicide"
As of 2.9, the month for this log reflects the correct month. (with january starting at 1)

There is also a hook that lets custom scripts.dll mods (e.g. server-side mods) get access to all f2/f3 chat that passes through the server.
You create a function of the form
void Chat_Hook(int PlayerID,TextMessageEnum Type,const wchar_t *Message) (note the change from char * in 2.1 to wchar_t * in 2.2)
Then you pass the function to AddChatHook (defined in engine.h)
If the chathook is called Chat_Hook, put the line AddChatHook(Chat_Hook); somewhere in your

code. (e.g. somewhere that is called on startup)
Then, the function gets called everytime f2/f3 chat passes through the server.
PlayerID is the player ID of the player that sent the chat.
Type is PrivateMessage for player-to-player private message, TeamMessage for team mesasge and PublicMessage for everyone message.
Message is the message itself. If you need to save the message data for later use, dont save the pointer passed into your chat
hook function, copy the data somewhere else.
You can only have one chat hook function registered at any one time. Also, if you want to have no chat hook at all registered, pass NULL to AddChatHook.
This works on the server regardless of if clients have bhs.dll

This functionality is quite useful if you want to make new !xxx commands, especially since you can use the player ID to verify that the person using the command
is authorized to use it. It also means you can do all the things that you would otherwise need new console commands to do.

There is a hook that lets custom scripts.dll mods see all host messages (i.e. all messages done via message, tpage and ppage)
You create a function of the form
void Host_Hook(int PlayerID,TextMessageEnum Type,const char *Message)
Then you pass the function to AddHostHook (defined in engine.h)
If the hosthook is called Host_Hook, put the line AddHostHook(Host_Hook); somewhere in your code. (e.g. somewhere that is called on startup)
Then, the function gets called when the message, ppage or tpage console commands are used on the server.
Type is PrivateMessage for private message, TeamMessage for team message, PublicMessage for everyone message
Type TMSGCommand is for the TMSG console command.
Message is the message itself. If you need to save the message data for later use, dont save the pointer passed into your host
hook function, copy the data somewhere else.
if Type is PrivateMessage, PlayerID is the player ID of the target
if Type is TeamMessage, PlayerID is the team its being sent to (0 = nod, 1 = GDI)
if Type is PublicMessage, PlayerID should be ignored
if Type is TMSGCommand, PlayerID is the player ID of the sender
You can only have one host hook function registered at any one time. Also, if you want to have no host hook at all registered, pass NULL to AddHostHook.
This works on the server regardless of if clients have bhs.dll

There is also an engine call called GetCurrentMusicTrack that returns the same thing as the SONG console command.
There is also an engine call called GetBHSVersion that returns the version of bhs.dll installed on the server.

If the client is running bhs.dll >= 2.2, then when they join, a message will be sent to the server, causing the same output as for the VERSION console command to be output for that player.
The VERSION console command will also work on these players as it does on players with < 2.2

There is also code to fix the invisible harvester bug (where if you have an airstrip, go low power and your harvester is destroyed, it spawns invisible).
However, the downside is that you can see the wheels/treads of the vehicle sticking out of the cargo plane when it flies in.
What normally happens is that the vehicle is created and made invisible and the cinematic is started. When the vehicle is dropped off, it is made visible again.
However, in the bug case, it is not made visible again. The fix stops it from being made invisible in the first place.

The linux version of BHS.DLL also contains a fix to make the IP address display properly in the PLAYER_INFO console command.
All of these last few items work on the server regardless of if the client has bhs.dll installed.

Also, there is now new code in bhs.dll to make the infantry death sound and powerup collection sound play on the client as well as the server.
You need bhs.dll on the server and on the client for this to work.

There is also code in bhs.dll to disable certain network interfaces that could be used by a cheater to run console commands on the server from the client, kill
someone from the client, steal someone elses money and other undesirable things.

And there is code to block people with an invalid nickname.
If the nickname of the new player matches any of the following, the player is denied a connection and a console message listing the IP address of the denied player is displayed:
"Player with invalid nickname blocked, player IP was 1.2.3.4"
The player will then be denied connection.
Blocked nicknames are those with:
Nickname length = 0
Nickname length > 35
Nickname = Hostname
Nickname has non-ascii characters (i.e. below ' ' or above '~')
Nickname is all spaces
Nickname matches a name already in use on the server
Last character of the nickname is a space
Nickname contains more than one space next to each other

This code replaces the existing bandtest.dll fix (so you dont need it anymore, I suggest removing it in case it conflicts).
Credit goes to Slent_Kane for releasing the first working fix for the nickname exploits.

These 2 fixes dont require bhs.dll on the client and will work on the Linux FDS (unlike the bandtest.dll fix)

There is new code on windows (client, FDS, server, whatever) that, when renegade crashes, will dump a new crashdump.txt file.
The old _except.txt stuff is gone as is any apperance of the "renegade crashed at address xxx" windows crash dialogs that used to appear.

Now, if it crashes, it will exit silently and spit out a crashdump.txt file in your renegade folder.
This file contains the following:
The address the exception occured at (including which module and which segment of that module the address resides in)
The address being accessed when the crash happens (including if it was trying to read from that address or write to it)
Whether the crash happened with the game client or the FDS
The contents of the CPU registers at the time the crash happened
The contents of the FPU registers at the time the crash happened
The bytes currently in memory at the address the crash happened at
The current version of bhs.dll (i.e. the same thing printed by the sversion/version console commands)
The current version of windows (specifically all the return values from GetVersionEx)
The current time and date (great for matching crashdump.txt files with events in a server log file)
If appropriate, the current map, mod package, player count and time remaining
A complete list of all modules loaded into the renegade process space along with the address they are loaded at and their CRC32
And a complete dump of the processor stack at the time the crash happened (along with details of which module and which segment of that module the address resides in)

as of 3.2 crashdump files are generated with names like crashdump1.txt, crashdump2.txt etc ala screenshots.

There is also a hook that lets custom scripts.dll mods (e.g. server-side mods) hook whenever a player joins the server
You create a function of the form
void Player_Join_Hook(int PlayerID,const char *PlayerName)
Then you pass the function to AddPlayerJoinHook (defined in engine.h)
Then, the function gets called everytime a player joins the server
PlayerID is the player ID of the player that just joined
PlayerName is the player name of the player that just joined.
I dont know if the player has a valid GameObject at this point, you could try calling Get_GameObj and finding out if it returns something other than 0.
If you need to save the player name for later use, dont save the pointer passed into your hook function, copy the data somewhere else.
You can only have one player join hook function registered at any one time. Also, if you want to have no hook at all registered, pass NULL to AddPlayerJoinHook.
This works on the server regardless of if clients have bhs.dll

There is also a hook that lets custom scripts.dll mods (e.g. server-side mods) hook on level load
You create a function of the form
void Load_Level_Hook()
Then you pass the function to AddLoadLevelHook (defined in engine.h)
Then, the function gets called when a map is loaded.
All the GameObjects are valid at this point so functions like Find_Soldier_Factory (for example) will work and return valid data.
You can only have one load level hook function registered at any one time. Also, if you want to have no hook at all registered, pass NULL to AddLoadLevelHook.

This works on the server regardless of if clients have bhs.dll

There is also a hook that lets custom scripts.dll mods (e.g. server-side mods) hook on gameover
You create a function of the form
void Game_Over_Hook()
Then you pass the function to AddGameOverHook (defined in engine.h)
Then, the function gets called when the game ends.
Use such things as The_Game()->WinnerID, The_Game()->WinType,Get_Team_Score() & Tally_Team_Size() to retrieve information related to who won etc
You can only have one game over hook function registered at any one time. Also, if you want to have no hook at all registered, pass NULL to AddGameOverHook.
This works on the server regardless of if clients have bhs.dll

There is new code that lets you change the color used for the teams (i.e. red for nod and yellow for gdi)
You can also change the public and private message colors.
Note that the hud.ini file is parsed on the server too (for the colored message send commands and the get color engine calls).
This requires bhs.dll on the client.
You create a hud.ini file that looks something like this
[General]
NodHouseRed=255
NodHouseGreen=0
NodHouseBlue=0
GDIHouseRed=255
GDIHouseGreen=204
GDIHouseBlue=0
PrivateMessageRed=0
PrivateMessageGreen=0
PrivateMessageBlue=255
PublicMessageRed=255
PublicMessageGreen=255
PublicMessageBlue=255

(for reference, the numbers above are the default numbers).
This ini file will be read by bhs.dll (if it exists) and the colors defined therein (if any values are defined) will be used instead of the default colors.
As the name suggests, hud.ini will be used for more in-game user interface related things in the future Smile

There is also code to disable the GameSpy CD check in the game client, WFDS and LFDS. This was done because:
A.The The First Decade keys are not in the GameSpy database.
B.We needed it for the LFDS WOL code
and C.XWIS doesnt check CD keys anyway.

There is a code change that should make messages "xxx left game", "xxx joined game" and "xxx bought a vehicle" display 100% of the time.

There is a code change to the TEAM and TEAM2 console commands so that they will disarm all C4 of the player before switching teams.
The SVERSION console command is no longer host only and will print the version of bhs.dll installed on the client if used on the client.
There is new code in the keyhooks and keycfg.exe

There are new engine calls Get_Vehicle_Limit and Set_Vehicle_Limit which do just as they say. Set_Vehicle_Limit requires bhs.dll on the client.

There is a new feature that will display a texture on a clients machine (e.g. a set of instructions).
The engine calls Set_Info_Texture and Clear_Info_Texture are used for this.
The texture should be made exactly the same as for the Scope feature (and can have alpha transparency if you
only want your texture to appear in part of the screen)
This feature requires bhs.dll on the client

There are new engine calls Send_Message_Player and Send_Message which send a colored message to either a specific player or to everyone.
The message will appear in the same box as messages like "Unit Ready".
Each message sent will appear on its own line (i.e. there is an implicit newline) and has one RGB color associated with it.
This feature requires bhs.dll on the client.

There is a new hook that will get called any time the "The Version of player x is y" string is displayed.
The hook is passed the player ID and the version number.
define the hook as void VersionHook(int PlayerID,float Version) and call AddVersionHook to add it.
As with the other hooks, pass zero if you dont want any hook installed.

There is a new engine call Set_Wireframe_Mode which will set the wireframe mode for the level.
Valid values are:
1 = point
2 = wireframe
3 = solid
This stays in effect untill changed again.
This feature requires bhs.dll on the client
Also, note that this call changes ALL the graphics into wireframe including the HUD, text strings, dialog boxes (e.g. PTs), menus etc.
As of 2.8, this changes all the 3d objects on the level (e.g. your gun/character, enemies, map, buildings, powerups etc etc) but not the 2D UI (hud, dialogs, menus etc)

There is a new engine call Load_New_HUD_INI which loads a new hud ini for a given player
See below for details of
This feature requires bhs.dll on the client.

There is a new engine call Remove_Weapon which removes a weapon from a player. This used to be in engine.cpp but was moved to bhs.dll

as it turns out that it requires sending data over the network.
This feature requires bhs.dll on the client.

There is a new mechanisim to change the PT data at runtime.
How it works is this:
As of 2.8, the PT data is no longer saved or restored as it was found that the game reloads the PT data every map load.
This means the problems with tempped PT presets are gone.
First, you change the PT data with the engine calls in engine.cpp and then call Update_PT_Data to send the new PT data to all players (calling Update_PT_Data also sets the flag that indicates that it is modified)
When a player joins, if it has been changed since the map loaded, it is sent to the player.
This feature requires bhs.dll on the client.
You will want to use engine calls like Set_Enlisted, Set_Beacon, Set_Refill, Set_Preset, Set_Alternate, Disable_Enlisted and Disable_Preset to change the PT data.
Look for these engine calls in the scripts.dll codebase to see examples of how they are used.

There is new functionality on linux to send the special packet required for LFDS WOL (which used to be sent by the WOLSEND command)

Also, hud.ini now contains features to totally customize the HUD.
Any time you are setting a screen position (e.g. HealthXPos), if you pass a negative number, it will move that many units in from the bottom right of the screen, otherwise it moves from the top left of the screen)
This is so you can build HUDs with items in bottom and left corners whilst remaining independant of screen resolution.
Unless specified otherwise, all entries go in the [General] section of hud.ini
Also, any time you specify a texture, always use .tga on the end even if the texture is a dds file.
Where it specifies a font number from stylemgr.ini, 0 is FONT_TITLE and so on. Adding new fonts is not possible.

Firstly, you can define custom colors that will be used later on in hud.ini
You set ColorCount to specifiy how many colors to add.
Then you create sections labeled [Color1] and so on on up. Each section has entries Red, Green and Blue to specify the color (values go from 0 to 255)
In all the entries below, anytime a color is mentioned, it is one of these colors specified here.
Color zero is the default color and is black.

You can define customization for the number that shows your current health.
Set HealthEnabled=true to turn it on (and turn the default westwood logic off)
Set HealthVisible=false to completly hide the number otherwise it will be drawn
Set HealthXPos and HealthYPos to specify where on the screen to draw the number.
Set HealthXOffset and HealthYOffset to specify the health offset (basicly, this number is multiplied by the current % health the player has and added to the health position when drawing)
Set HealthFont to a texture name (i.e. a texture similar to FONT12x16.TGA) to specify the font to use.
Set HealthColor to a color index to specify the color to use.
There is also a way (which overrides HealthColor) to set colors depending on how much health

you have)
HealthColorCount specifies how many color entries there are.
Then you create sections like [HealthColor0] etc to specify the colors.
Each section has keywords Color which is the color index to use and Value which is the percentage of health to start drawing at.
The sections must be organized so that the smallest value is section 0 and so on on up.

You can define customization for the number that shows your current shield.
Set ShieldEnabled=true to turn it on (and turn the default westwood logic off)
Set ShieldVisible=false to completly hide the number otherwise it will be drawn
Set ShieldXPos and ShieldYPos to specify where on the screen to draw the number.
Set ShieldXOffset and ShieldYOffset to specify the shield offset (basicly, this number is multiplied by the current % shield the player has and added to the shield position when drawing)
Set ShieldFont to a texture name (i.e. a texture similar to FONT12x16.TGA) to specify the font to use.
Set ShieldColor to a color index to specify the color to use.
There is also a way (which overrides ShieldColor) to set colors depending on how much shield you have)
ShieldColorCount specifies how many color entries there are.
Then you create sections like [ShieldColor0] etc to specify the colors.
Each section has keywords Color which is the color index to use and Value which is the percentage of shield to start drawing at.
The sections must be organized so that the smallest value is section 0 and so on on up.

You can define customization for the number that shows the current bullets in your gun
Set BulletEnabled=true to turn it on (and turn the default westwood logic off)
Set BulletVisible=false to completly hide the number otherwise it will be drawn
Set BulletXPos and BulletYPos to specify where on the screen to draw the number.
Set BulletFont to a texture name (i.e. a texture similar to FONT12x16.TGA) to specify the font to use.
Set BulletColor to a color index to specify the color to use.

You can define customization for the number that shows the current bullets in your "backpack"
Set ClipEnabled=true to turn it on (and turn the default westwood logic off)
Set ClipVisible=false to completly hide the number otherwise it will be drawn
Set ClipXPos and ClipYPos to specify where on the screen to draw the number.
Set ClipFont to a texture name (i.e. a texture similar to FONT12x16.TGA) to specify the font to use.
Set ClipColor to a color index to specify the color to use.

You can define customization for the weapon name.
Set WeaponEnabled=true to turn it on (and turn the default westwood logic off)
Set WeaponVisible=false to completly hide the name otherwise it will be drawn
Set WeaponVisibleNonVehicle=false to hide the name whilst not in a vehicle
Set WeaponXPos and WeaponYPos to specify where on the screen to draw the name.
Set WeaponFont to a font number from stylemgr.ini
Set WeaponColor to a color index to specify the color to use.

You can define customization for the weapon image.
Set WeaponImageEnabled=true to turn it on (and turn the default westwood logic off)
Set WeaponImageVisible=false to completly hide the image otherwise it will be drawn
Set WeaponImageVisibleNonVehicle=false to hide the image whilst not in a vehicle (i.e. you see the steering wheel, gun and seat icons but not the weapon icon)
Set WeaponImageXPos and WeaponImageYPos to specify where on the screen to draw the name.
Set WeaponImageColor to a color index to specify the color to use.
Set WeaponImageVehicleColor to a color index to specify the color to use when inside a vehicle.

You can hide the names of enemy soldiers.
Set HideEnemySoldiers=true to enable this feature
Then, set HideExceptionCount to define the count of how many exceptions (exceptions are for presets like spies etc that you want to keep visible)
Set HideException0 and so on to the preset IDs of the presets you want to hide

You can define arbitrary textures to be drawn on the screen. (this is for the general parts of the HUD)
Set TextureCount to the count of how many textures you want
Create a section e.g. [Texture0] for each texture
Then, under that section, set TextureName to the texture name.
Set QuadCount to the count of how many rectangles to draw with this texture
Set Quad0Color (and on up) to the color for that rectangle
Set Quad0XPos (and on up) to the X screen position for that rectangle
Set Quad0YPos (and on up) to the Y screen position for that rectangle
Quad0Top, Quad0Left, Quad0Bottom & Quad0Right (and on up) define a rectangle within the texture which is what is drawn at the x,y screen position.

You can customize the health bar.
Set HealthBarEnabled=true to turn it on (and turn the default westwood logic off)
Set HealthBarVisible=false to completly hide the health bar otherwise it will be drawn
Set HealthBarEmptyVisible=false to hide the empty part of the healt bar otherwise it will be drawn
Set HealthBarColor to define the color index to use for the health bar
Set HealthBarEmptyColor to define the color index to use the for empty health bar
There is also a way (which overrides HealthBarColor) to set colors depending on how much health you have)
HealthBarColorCount specifies how many color entries there are.
Then you create sections like [HealthBarColor0] etc to specify the colors.
Each section has keywords Color which is the color index to use and Value which is the percentage of health to start drawing at.
The sections must be organized so that the smallest value is section 0 and so on on up.
Set HealthBarTexture to define the texture to use for the health bar
Set HealthBarXPos and HealthBarYPos to define the position on the screen of the health bar
HealthBarTop, HealthBarLeft, HealthBarBottom & HealthBarRight define a rectangle within the texture for the health bar
HealthBarEmptyTop & HealthBarEmptyLeft define the position of the empty health bar on the texture. The size is the same as the full health bar (i.e. when the health is not full to that spot)
Set HealthBarStyle=0 to enable a bar similar to the renegade health bar

Set HealthBarStyle=1 to enable a bar similar to the renegade shield strength bar
Set HealthBarDirection to determine which direction to draw for style 2 (1 is the way renegade draws it, 0 is the opposite direction)
Set HealthBarOffset to determine the offset to draw the icons at for style 2
Set HealthBarLength to set the length for the entire health bar for style 2
When the style is style 2, the HealthBarTop etc keywords define the icon that is to be used (e.g. like the shield in renegade)
The x and y position determines the position of the bar
The length determines the horizontal size and the height defined by the HealthBarTop etc keywords determine the vertical size

You can customize the shield bar.
Set ShieldBarEnabled=true to turn it on (and turn the default westwood logic off)
Set ShieldBarVisible=false to completly hide the shield bar otherwise it will be drawn
Set ShieldBarEmptyVisible=false to hide the empty part of the healt bar otherwise it will be drawn
Set ShieldBarColor to define the color index to use for the shield bar
Set ShieldBarEmptyColor to define the color index to use the for empty shield bar
There is also a way (which overrides ShieldBarColor) to set colors depending on how much shield you have)
ShieldBarColorCount specifies how many color entries there are.
Then you create sections like [ShieldBarColor0] etc to specify the colors.
Each section has keywords Color which is the color index to use and Value which is the percentage of shield to start drawing at.
The sections must be organized so that the smallest value is section 0 and so on on up.
Set ShieldBarTexture to define the texture to use for the shield bar
Set ShieldBarXPos and ShieldBarYPos to define the position on the screen of the shield bar
ShieldBarTop, ShieldBarLeft, ShieldBarBottom & ShieldBarRight define a rectangle within the texture for the shield bar
ShieldBarEmptyTop & ShieldBarEmptyLeft define the position of the empty shield bar on the texture. The size is the same as the full shield bar (i.e. when the shield is not full to that spot)
Set ShieldBarStyle=0 to enable a bar similar to the renegade health bar
Set ShieldBarStyle=1 to enable a bar similar to the renegade shield strength bar
Set ShieldBarDirection to determine which direction to draw for style 2 (1 is the way renegade draws it, 0 is the opposite direction)
Set ShieldBarOffset to determine the offset to draw the icons at for style 2
Set ShieldBarLength to set the length for the entire shield bar for style 2
When the style is style 2, the ShieldBarTop etc keywords define the icon that is to be used (e.g. like the shield in renegade)
The x and y position determines the position of the bar
The length determines the horizontal size and the height defined by the ShieldBarTop etc keywords determine the vertical size

Set HideWeaponBox=true to hide the background piece that is underneath the weapon information
Set HideInfoBox to hide the background piece that is over the other side (this covers everything except the radar that is not text including the health and shield bars)

You can also customize the compass (i.e. the N,E,S,W etc text)

Set EnableCompass=true to turn it on (and turn the default westwood logic off)
Set CompassVisible=false to completly hide the compass otherwise it will be drawn
Set CompassFont to a font number from stylemgr.ini
Set CompassColor to the color index to use when drawing the compass
Set CompassXPos and CompassYPos to define where to draw the compass

You can also customize the credits text.
Set EnableCredits=true to turn it on
Set CreditsXPos and CreditsYPos to define where to draw the credits display
Set CreditsFont to a font number from stylemgr.ini
Set CreditsColor to the color index to use when drawing the credits
Set CreditsStringID to a string ID from strings.tdb. This string should use %d at the spot where you want the credits number to go.
For example, "Current Credits is %d now" will replace the %d with the current credits

You can also customize the time remaining text (it will not be drawn if the level has unlimited time)
Set EnableTime=true to turn it on
Set TimeXPos and TimeYPos to define where to draw the time display
Set TimeFont to a font number from stylemgr.ini
Set TimeColor to the color index to use when drawing the time
Set TimeStringID to a string ID from strings.tdb. This string should use %s at the spot where you want the time to go.
For example, "There is %s remaining" will replace the %s with the current time formatted as hours:minutes:seconds just like the default display)

You can also display an icon for the health (like the + symbol in normal renegade)
Set HealthIconEnabled=true to turn it on
Set HealthIconColor to define the color index to use for the health icon
There is also a way (which overrides HealthIconColor) to set colors depending on how much health you have)
HealthIconColorCount specifies how many color entries there are.
Then you create sections like [HealthIconColor0] etc to specify the colors.
Each section has keywords Color which is the color index to use and Value which is the percentage of health to start drawing at.
The sections must be organized so that the smallest value is section 0 and so on on up.
Set HealthIconTexture to the texture to use for the health icon
Set HealthIconXPosition and HealthIconYPosition to the position to draw the health icon
HealthIconTop, HealthIconLeft, HealthIconBottom and HealthIconRight define the rectangle on the texture to use for the icon

You can also display an icon for the shield (like the + symbol in normal renegade)
Set ShieldIconEnabled=true to turn it on
Set ShieldIconColor to define the color index to use for the shield icon
There is also a way (which overrides ShieldIconColor) to set colors depending on how much shield you have)
ShieldIconColorCount specifies how many color entries there are.
Then you create sections like [ShieldIconColor0] etc to specify the colors.
Each section has keywords Color which is the color index to use and Value which is the

percentage of shield to start drawing at.
The sections must be organized so that the smallest value is section 0 and so on on up.
Set ShieldIconTexture to the texture to use for the shield icon
Set ShieldIconXPosition and ShieldIconYPosition to the position to draw the shield icon
ShieldIconTop, ShieldIconLeft, ShieldIconBottom and ShieldIconRight define the rectangle on the texture to use for the icon

Unlike the renegade code, you cant implement it so that the health/shield icon will flash when you have low health/shield

You can also customize the radar.
Set EnableRadar=true to turn it on and turn the default westwood logic off.
Set RadarRotate=true to make the radar rotate so that "up" is always the direction you are facing.
Set RadarRotate=false to make up always = north. (this is needed later on when I implement the scrolling map logic)
Set DrawStar=true to draw you (i.e. the current player) on the radar
Set DrawCompassLine=true to draw a line in the direction the player is facing (this is disabled if RadarRotate is set to true)
Set CompassLineColor to the color to use for the compass line
Set CompassLineWidth to the width for the compass line
Set RadarVisible=false to turn off the radar completly
Set RadarSize to the screen size in pixels to use for the radar (its always going to be a circle)
Set RadarWorldSize to the size in the game world that is to correspond to the radius of the radar circle
Set BackgroundColor to the color to use when drawing the background
Set BackgroundTexture to the texture to use for the background
Set BlipTexture to the texture to use for the radar blips
Set BackgroundTop and BackgroundLeft to the top left position on the background texture to draw from.
Set RadarX and RadarY to define the top left position of the square where the radar is to be drawn
Set ScrollingRadarMap=true to enable the scrolling map background feature (where you see a top-down view of the map under the radar blips which scrolls based on the location of the player)
Set RadarBlipnTop and RadarBlipnLeft (starting from RadarBlip1Top/RadarBlip1Left) to cover the UV positions for the radar blips
Blips should go from 1 to 5
The blips are always 8 pixels in size
Set RadarBlipColor0 and so on up to RadarBlipColor7 to define the blip colors
Blip zero is blank
Blip 1 is for humans (by default its a circle)
Blip 2 is for vehicles (by default its a triangle)
Blip 3 is for stationary objects (by default its a square)
Blip 4 is for objectives (by default its a star)
Blip 5 is for the bracket that is drawn when you are targeting an object
The blip colors are as follows:
Color 0 is red (Nod color)
Color 1 is gold (GDI color)
Color 2 is white

Color 3 is dark green
Color 4 is blue
Color 5 is green (and is used when the bracket is drawn)
Color 6 is light blue
Color 7 is purple
Right now, this code is not perfect, I plan to extend it later (for example, to make it draw objectives and radar markers)

How the scrolling map feature works:
If you turn it on with ScrollingRadarMap=true, you then need to set the settings for each map with something like JFW_Change_Radar_Map.
Basicly, it draws the radar background the same as the normal radar code with the exception of the texture coordinates for the background texture.
By default, the center of the map texture is assumed to match with 0,0 in the game world. Use the offsetx and offsety values to specify where on the texture (relative to the center) 0,0 in the game world is.
The scale field codes for how many pixels on the texture 1 unit in the game world is equal to.
Just play around with the offset and scale asnd see what looks right for your map.
As of 3.0, you can create an ini file called mapname.ini (e.g. C&C_Islands.ini) which contains the following tags in the [General] section
ScrollingMapTexture
ScrollingMapOffsetX
ScrollingMapOffsetY
ScrollingMapScale
These correspond to the parameters that get passed to Change_Radar_Map
This ini file goes on the client. If the map/server makes a call to Change_Radar_Map, it will override anything set by the ini file.

Set HideBottomText=true to disable the credits and time remaining dislay
Set HidePlayerList=true to hide the player and team list
Set HidePowerupIcons=true to hide the icons that show powerups (e.g. when you pick one up)
Set DisableCostMultiplier=true to disable the 2x cost multiplier that applies if the base power is down (both the display and the effect)
Set BuildTimeDelay=some number to change the vehicle build time multiplier (default is 2x) that applies if base power is down
Set VehicleOwnershipDisable=true to disable the effect where only the person who bought a vehicle can get into it for a while after its bought
Set UseExtraPTPages=true to enable use of the extra "hidden" PT pages (the ones you access with the alt key) as normal PT pages. This disables the extras console command, disables the laddered server check and enables the PT pages feature.
Set NewUnpurchaseableLogic=true to enable new logic for when vehicles and characters are being bought. This is needed if you wish to use Set_Can_Generate_Soldiers or Set_Can_Generate_Vehicles after the relavent buildings have gone down.
However, with it enabled, the text under the vehicle and character buttons will always say "unavailable", never "destroyed"
Also, vehicles dont become available again if the vehicle factory is offline (however, the vehicle purchase hook will be triggered) since there is no vehicle factory to deliver the vehicles
Set VehicleBuildingDisable=true to disable the code that makes the vehicle button show "building"

when a vehicle is bought from the weapons factory.
set the folowing keywords to change the colors associated with the various UI elements. Alpha is the indicator of how opaque that element is (FF = completly opaque). The value after each keyword is the default.
TitleColorAlpha=255
TitleColorRed=255
TitleColorGreen=255
TitleColorBlue=255
TitleHilightColorAlpha=255
TitleHilightColorRed=255
TitleHilightColorGreen=255
TitleHilightColorBlue=0
TitleShadowColorAlpha=255
TitleShadowColorRed=0
TitleShadowColorGreen=0
TitleShadowColorBlue=0
TextColorAlpha=255
TextColorRed=255
TextColorGreen=213
TextColorBlue=40
TextShadowColorAlpha=200
TextShadowColorRed=0
TextShadowColorGreen=0
TextShadowColorBlue=0
LineColorAlpha=255
LineColorRed=255
LineColorGreen=174
LineColorBlue=40
BkColorAlpha=40
BkColorRed=255
BkColorGreen=174
BkColorBlue=40
DisabledTextColorAlpha=140
DisabledTextColorRed=255
DisabledTextColorGreen=213
DisabledTextColorBlue=40
DisabledTextShadowColorAlpha=96
DisabledTextShadowColorRed=0
DisabledTextShadowColorGreen=0
DisabledTextShadowColorBlue=0
DisabledLineColorAlpha=128
DisabledLineColorRed=230
DisabledLineColorGreen=160
DisabledLineColorBlue=35
DisabledBkColorAlpha=30
DisabledBkColorRed=255
DisabledBkColorGreen=174
DisabledBkColorBlue=40

HilightColorAlpha=255
HilightColorRed=70
HilightColorGreen=70
HilightColorBlue=70
TabTextColorAlpha=255
TabTextColorRed=255
TabTextColorGreen=255
TabTextColorBlue=255
TabGlowColorAlpha=255
TabGlowColorRed=16
TabGlowColorGreen=10
TabGlowColorBlue=0
DialogTextTitleColorAlpha=255
DialogTextTitleColorRed=255
DialogTextTitleColorGreen=255
DialogTextTitleColorBlue=36
DialogTextTitleGlowColorAlpha=255
DialogTextTitleGlowColorRed=14
DialogTextTitleGlowColorGreen=0
DialogTextTitleGlowColorBlue=0
MenuHiliteColorAlpha=255
MenuHiliteColorRed=0
MenuHiliteColorGreen=0
MenuHiliteColorBlue=0
MerchandiseTextColorAlpha=0
MerchandiseTextColorRed=255
MerchandiseTextColorGreen=255
MerchandiseTextColorBlue=255

Set DisableMenuCtrlGlow=true to disable the glow around the main menu items
Set ListColumnColorEnabled=true to enable changing of the list column colors.
Set ListColumnColorRed, ListColumnColorBlue and ListColumnColorGreen to change the list column colors from the default white.

Set Unsquishable=true to turn on the new unsquishable logic
Then you need to set UnsquishableArmor=<armor number from list in armor.ini> to set the special armor.
For example, ShieldKevlar is 15
With this enabled, the game wont squish any infantry unit that has this special armor type.
You can also set UnsquishableArmor2, UnsquishableArmor3 and UnsquishableArmor4 for up to 4 different types of unsquishable armor.

Set StealthRenderStateChangeEnabled=true to turn on the new stealth texture effect logic.
EDIT: Due to difficulties in making this feature work properly, it has been removed.
You can use a custom shader through shaders.dll instead

The following keywords apply to the sidebar logic
GDIUpArrowTexture is the texture for the up arrow button for GDI

GDIDownArrowTexture is the texture for the down arrow button for GDI
GDIBackgroundTexture1 is the upper half of the background texture for GDI
GDIBackgroundTexture2 is the lower half of the background texture for GDI
NODUpArrowTexture is the texture for the up arrow button for Nod
NODDownArrowTexture is the texture for the up arrow button for Nod
NODBackgroundTexture1 is the upper half of the background texture for Nod
NODBackgroundTexture2 is the lower half of the background texture for Nod
RefillLimit is the refill limit in seconds
AlternateSelectEnabled determines if the special alternate selection logic is enabled
GDIAlternateSelectTexture1
GDIAlternateSelectTexture2
GDIAlternateSelectTexture3
GDIAlternateSelectTexture4
These 4 are the 4 textures for the alternate selection buttons for GDI
NODAlternateSelectTexture1
NODAlternateSelectTexture2
NODAlternateSelectTexture3
NODAlternateSelectTexture4
These 4 are the 4 textures for the alternate selection buttons for Nod
Note that the 4 background textures can have alpha transparency
SidebarSoundsEnabled=true. Set this to have new sounds for the sidebar purchasing.
SidebarRefillSound Set this to the sound to use when refilling.
SidebarInfantrySound Set this to the sound to use when buying infantry
SidebarVehicleSound Set this to teh sound to use when buying vehicles.
See below for full details of the working of the sidebar.

ModReg is the mod specific registry string for the data send function.
Set VersionReg to the path to a registry string (e.g. Software\Westwood\Renegade) for the version registry string change feature.
Set WOLUrlReg to the path to a registry string (e.g. WOLSettings\URL) for the WOL URL registry change feature.
Set DisableKillMessages=true to turn off the default kill messages

Set ShaderCheckMaterial=false to match against the texture name when matching shader names. Set it to true (the default) to match against material names

The following keywords need to go on hud.ini on the server
NodHouseRed
NodHouseGreen
NodHouseBlue
GDIHouseRed
GDIHouseGreen
GDIHouseBlue
PrivateMessageRed
PrivateMessageGreen
PrivateMessageBlue
PublicMessageRed
PublicMessageGreen

PublicMessageBlue
DisableCostMultiplier
BuildTimeDelay
VehicleOwnershipDisable
VehicleBuildingDisable
Unsquishable
UnsquishableArmor
UnsquishableArmor2
UnsquishableArmor3
UnsquishableArmor4
DisableKillMessages
The others are client-side only (but all of the above go on the client as well as the server).
Any keywords found in hud.ini on the server that the server doesnt read are ignored. (obviously if the server is also a client, everything gets read Smile

The following keywords can be changed in the runtime ini files:
HealthColor
the Color keyword in [HealthColor] sections
ShieldColor
the Color keyword in [ShieldColor] sections
BulletColor
ClipColor
WeaponColor
CompassColor
CreditsColor
TimeColor
WeaponImageColor
WeaponImageVehicleColor
The TextureName keyword in [Texture] sections
The QuadnColor keywords in [Texture] sections
HealthBarColor
HealthBarEmptyColor
The Color keyword in [HealthBarColor] sections
HealthBarTexture
ShieldBarColor
ShieldBarEmptyColor
The Color keyword in [ShieldBarColor] sections
ShieldBarTexture
HealthIconColor
The Color keyword in [HealthIconColor] sections
HealthIconTexture
ShieldIconColor
The Color keyword in [ShieldIconColor] sections
ShieldIconTexture
BackgroundColor
CompassLineColor
BackgroundTexture
BlipTexture

RadarBlipColorn keywords

The runtime ini files have the exact same layout as hud.ini
All keywords in the runtime ini files that arent read at runtime are ignored.

The best way to understand the hud.ini code is to play with it and see what works/looks good.

There are patches that are applied that will stop several possible server exploits.

There is a patch that should make disabling the radar via the Com Center building or via the Enable_Base_Radar engine call work properly.

There is a new engine call, Get_Build_Time_Multiplier which gets the current build time multiplier for a team (either 1.0 or whatever it is set to in the ini file depending on whether the power is down)

There is a new engine call, Change_Radar_Map which lets you change the current settings (scale, offsetx, offsety, texture) for the scrolling radar map.

There is also a hook that hooks the powerup purchase event (i.e. beacon purchase).
You create a function of the form
int Powerup_Purchase_Hook(BaseControllerClass *base, GameObject *purchaser, unsigned int cost, unsigned int preset)
Then you pass the function to AddPowerupPurchaseHook (defined in engine.h)
To remove the hook, pass the return value from AddPowerupPurchaseHook to RemovePowerupPurchaseHook
Then, the function gets called everytime someone buys a beacon from the beacon buy button.
base is the BaseControllerClass instance that belongs to the team doing the purchasing.
purchaser is the soldier object doing the purchasing
cost is the cost of the item being purchased
preset is the preset ID of the item being purchased
You return one of the following values to determine what the code does:
-1 = allow the next hook or default code to run
Other values cause messages to be output to the player doing the purchasing:
0 = "Purchase request granted."
1 = "Transaction pending."
2 = "You have insufficient funds for this purchase."
3 = "The factory is not presently available."
4 = "This item is not presently in stock."
You can only have one powerup purchase hook function registered at any one time. Also, if you want to have no Powerup Purchase Hook at all registered, pass NULL to AddPowerupPurchaseHook.
This works on the server regardless of if clients have bhs.dll

There is also a hook that hooks the vehicle purchase event
You create a function of the form
int Vehicle_Purchase_Hook(BaseControllerClass *base, GameObject *purchaser, unsigned int cost, unsigned int preset,const char *data)

Then you pass the function to AddVehiclePurchaseHook (defined in engine.h)
You also pass in another piece of data, it can be whatever you like. This piece of data will be passed back to the hook.
To remove the hook, pass the return value from AddVehiclePurchaseHook to RemoveVehiclePurchaseHook
Then, the function gets called everytime someone buys a vehicle from the PT.
base is the BaseControllerClass instance that belongs to the team doing the purchasing.
purchaser is the soldier object doing the purchasing
cost is the cost of the item being purchased
preset is the preset ID of the item being purchased
You return one of the following values to determine what the code does:
-1 = allow the next hook or default code to run
Other values cause messages to be output to the player doing the purchasing:
0 = "Purchase request granted."
1 = "Transaction pending."
2 = "You have insufficient funds for this purchase."
3 = "The factory is not presently available."
4 = "This item is not presently in stock."
You can only have one vehicle purchase hook function registered at any one time. Also, if you want to have no vehicle purchase hook at all registered, pass NULL to AddVehiclePurchaseHook.
This works on the server regardless of if clients have bhs.dll

There is also a hook that hooks the character purchase event
You create a function of the form
int Character_Purchase_Hook(BaseControllerClass *base, GameObject *purchaser, unsigned int cost, unsigned int preset,const char *data)
Then you pass the function to AddCharacterPurchaseHook (defined in engine.h)
You also pass in another piece of data, it can be whatever you like. This piece of data will be passed back to the hook.
To remove the hook, pass the return value from AddCharacterPurchaseHook to RemoveCharacterPurchaseHook
Then, the function gets called everytime someone buys a character from the PT. (including the free characters)
base is the BaseControllerClass instance that belongs to the team doing the purchasing.
purchaser is the soldier object doing the purchasing
cost is the cost of the item being purchased
preset is the preset ID of the item being purchased
You return one of the following values to determine what the code does:
-1 = allow the next hook or default code to run
Other values cause messages to be output to the player doing the purchasing:
0 = "Purchase request granted."
1 = "Transaction pending."
2 = "You have insufficient funds for this purchase."
3 = "The factory is not presently available."
4 = "This item is not presently in stock."
This works on the server regardless of if clients have bhs.dll

There are also a hook that lets you monitor the purchasing of characters.

You create a function of the form
void Character_Purchase_Monitor_Hook(BaseControllerClass *base, GameObject *purchaser, unsigned int cost, unsigned int preset,unsigned int purchaseret,const char *data)
Then you pass the function to AddCharacterPurchaseMonHook (defined in engine.h)
You also pass in another piece of data, it can be whatever you like. This piece of data will be passed back to the hook.
To remove the hook, pass the return value from AddCharacterPurchaseMonHook to RemoveCharacterPurchaseMonHook
Then, the function gets called everytime someone buys a character from the PT. (including the free characters)
base is the BaseControllerClass instance that belongs to the team doing the purchasing.
purchaser is the soldier object doing the purchasing
cost is the cost of the item being purchased
preset is the preset ID of the item being purchased
purchaseret is the final return value from the purchase code which dictates what message will be printed (and what action, if any, will be taken).
0 = "Purchase request granted."
1 = "Transaction pending."
2 = "You have insufficient funds for this purchase."
3 = "The factory is not presently available."
4 = "This item is not presently in stock."
This works on the server regardless of if clients have bhs.dll

There are also a hook that lets you monitor the purchasing of powerups.
You create a function of the form
void Powerup_Purchase_Monitor_Hook(BaseControllerClass *base, GameObject *purchaser, unsigned int cost, unsigned int preset,unsigned int purchaseret,const char *data)
Then you pass the function to AddPowerupPurchaseMonHook (defined in engine.h)
You also pass in another piece of data, it can be whatever you like. This piece of data will be passed back to the hook.
To remove the hook, pass the return value from AddPowerupPurchaseMonHook to RemovePowerupPurchaseMonHook
Then, the function gets called everytime someone buys a beacon from the beacon buy button.
base is the BaseControllerClass instance that belongs to the team doing the purchasing.
purchaser is the soldier object doing the purchasing
cost is the cost of the item being purchased
preset is the preset ID of the item being purchased
purchaseret is the final return value from the purchase code which dictates what message will be printed (and what action, if any, will be taken).
0 = "Purchase request granted."
1 = "Transaction pending."
2 = "You have insufficient funds for this purchase."
3 = "The factory is not presently available."
4 = "This item is not presently in stock."
This works on the server regardless of if clients have bhs.dll

There are also a hook that lets you monitor the purchasing of vehicles.
You create a function of the form

void Vehicle_Purchase_Monitor_Hook(BaseControllerClass *base, GameObject *purchaser, unsigned int cost, unsigned int preset,unsigned int purchaseret,const char *data)
Then you pass the function to AddVehiclePurchaseMonHook (defined in engine.h)
You also pass in another piece of data, it can be whatever you like. This piece of data will be passed back to the hook.
To remove the hook, pass the return value from AddVehiclePurchaseMonHook to RemoveVehiclePurchaseMonHook
Then, the function gets called everytime someone buys a vehicle from the PT.
base is the BaseControllerClass instance that belongs to the team doing the purchasing.
purchaser is the soldier object doing the purchasing
cost is the cost of the item being purchased
preset is the preset ID of the item being purchased
purchaseret is the final return value from the purchase code which dictates what message will be printed (and what action, if any, will be taken).
0 = "Purchase request granted."
1 = "Transaction pending."
2 = "You have insufficient funds for this purchase."
3 = "The factory is not presently available."
4 = "This item is not presently in stock."
This works on the server regardless of if clients have bhs.dll

There is a new engine call Set_Currently_Building which sets a flag for a team.
If this flag is set to true, the PT code displays "building" on the vehicle button (even if the vehicle factory is not currently building).

There is a new engine call Is_Currently_Building which returns the value set by Set_Currently_Building for that team.
These last 2 engine calls are intended to be used along with other features in bhs.dll and scripts.dll to implement working naval yards, helipads, or other buildings
such that you can buy vehicles through the PT and have them appear at other places other than the normal WF/airstrip/whatever

There is a new feature which I call RenLogMon. Basicly, what you do is to open a UDP listening socket on any internet connected machine and on any port.
Then you pass to the RLMON console command a string like 1.2.3.4:5 (ip and port) to enable logging.
Then, anytime data goes through the interface rlmon uses (which will catch console command output, chat stuff, pretty much everything that is printed to the FDS console), you get sent that data.
You can use RLMONOFF if you want to stop recieveing data logs.
RenLogMon is for dedicated servers only.
The data is sent as a raw UDP packet and is a null terminated string of variable length.
You can only have one mointor registered at once.
As of 2.9, there is code to display more output when you enable/disable renlogmon so that its easier to see whats going on with it.
Also, there is a new config file rlmon.cfg. Put the same string into it as would be passed to the rlmon console command and it will be read at server startup.

Also, there is a hook that lets you get the same text as rlmon gives you.
You create a function of the form
void Console_Output_Hook(const char *output)
Then you pass the function to AddConsoleOutputHook (defined in engine.h)
The function gets called at the same time as data is sent to RLMON (if it is registered)
You can only have one console output hook function registered at any one time. Also, if you want to have no hook at all registered, pass NULL to AddConsoleOutputHook.
This works on the server regardless of if clients have bhs.dll
Like RenLogMon, this functionality is for dedicated servers only.

There is new logic to change the reticle textures.
Call Set_Reticle_Texture1 to change the HD_reticle.tga texture and Set_Reticle_Texture2 to change the HD_reticle_hit.tga texture. Both of them take a GameObject that
is the object of the player you want to change the texture for.
When you set these textures, they will stay set untill set to something else.
When passing the texture, put .tga on the end even if it is a .dds file.
This requires bhs.dll on the client

There are some new engine calls to change some fog related settings
Set_Fog_Color changes the fog color for everyone and takes red, green, blue. Pass 256,256,256 to re-enable the normal fog color logic
Set_Fog_Color_Player change the fog color for a player and takes a GameObject plus red, green, blue. Pass 256,256,256 to re-enable the normal fog color logic
Set_Fog_Mode changes the fog mode for everyone. Use the FOG_xxx constants in engine.h FOG_LINEAR is the default mode.
Set_Fog_Mode_Player changes the fog mode for a player and takes a GameObject plus the mode.
Set_Fog_Density changes the fog density for everyone
Set_Fog_Density_Player changes the fog density for a player and takes a GameObject plus the density.
Set_Fog_Density and Set_Fog_Density_Player only have an effect when the fog mode is FOG_EXP or FOG_EXP2.
Values go from 0.0 to 1.0 with 1.0 being the most dense
Commands->Set_Fog_Range and Set_Fog_Range_Player only have an effect when the fog mode is FOG_LINEAR.
If you call Set_Fog_Color_Player, Set_Fog_Mode_Player or Set_Fog_Density_Player, that overrides whatever was set globally.
Conversly, calling the global functions changes everyone including those who have per-player fog settings set.
You should only call the fog functions here if Commands->Set_Fog_Enable or Set_Fog_Enable_Player has been called to enable fog first.
You should call Set_Fog_Color or Set_Fog_Color_Player with 256,256,256 before you call Set_Fog_Enable or Set_Fog_Enable_Player to disable fog.
You should also call Set_Fog_Mode or Set_Fog_Mode_Player to restore fog mode to FOG_LINEAR before the end of the map
This requires bhs.dll on the client

There is a new feature that lets you set the player limit at runtime.

You set the player limit via svrcfg_cnc.ini as normal.
Then you can use the plimit console command at runtime to change the player limit to anything that is <= that limit.
This changes what is displayed by the Game_Info console command as well as what is displayed by GameSpy and WOL in the server browsers.
And it will change bump a player with "server full" if they join when the current players is >= the max players set with plimit.
This feature will allow you to (for example) "reserve" player slots and open them up with plimit when you want to use them.
This will work even without bhs.dll on the client

There is a fix for the "start button" XWIS bug.

There is a new feature that sends the crc32 of hud.ini to the server on client startup. This is so that people
cant change it and cheat (e.g. enabling enemy names again when they have been disabled)
There is a hook that gets given the CRC and the player ID of the player whos hud.ini has that CRC.
You create a function of the form
void CRC_Hook(unsigned int crc, int ID)
Then you pass the function to AddCRCHook (defined in engine.h)
You can only have one hud.ini CRC hook function registered at any one time. Also, if you want to have no hook at all registered, pass NULL to AddCRCHook.

There are 2 new engine calls, Change_Time_Remaining and Change_Time_Limit which change the time remaining and time limit for the current game.
They require bhs.dll on the client.

There is a new feature that displays the chat history for the client (same as the PT does)
When the "chathistory" logical key (set in keys.cfg and used via my keyboard hook code) is pressed, a fullscreen window is opened showing the current chat history.
Pressing escape will close this window (just like any dialog).
Pressing F5 will refresh the chat history (unlike the PT, new chat is not automatically added to it)

There is a new feature, the sidebar. To activate it, call Display_GDI_Sidebar or Display_NOD_Sidebar which will display the sidebar for the given player.
The first button is the refill button then 3 infantry buttons and 4 vehicle buttons.
When the sidebar is created, if alternate selection is off, the alternate selection buttons are hidden otherwise the textures for them are loaded.
Also, the background and arrow textures are loaded.
And the refill button has its data set.
When displaying the sidebar buttons, the strings set in the PT data are not displayed.
If you want strings, you need to put them on the icon itself.
However, if the strings are set to string IDs 7265, 7263 or 9724, the PT icon is disabled.
Also, if the string is set to string ID 12574, the PT icon will be ignored/hidden.
If the preset ID is zero, that item is not displayed at all.
If the alternate select feature is not enabled, the normal yellow arrow alternate feature works.
Otherwise, the special alternate select feature is used.

If the flag set by Set_Currently_Building is set, the 4 vehicle buttons are disabled.
If less than <refilllimit> seconds have elapsed since the last refill by this player, the refill button is disabled.
If the special alternate select feature is enabled, the currently selected button is disabled (as a visual indicator of which alternate is selected)
When a sidebar purchase button that is not disabled is clicked, it is highlighted. When it is double clicked, the relavent purchase is made.
Pressing the up and down arrows scrolls the relavent list.
Pressing the alternate buttons (if the feature is enabled), changes the current alternate.
When items are purchased, if the sidebar sounds feature is enabled, the relavent sounds are played.

The following keys can be pressed whilst the sidebar is active:
Escape to cancel the dialog and close it.
Down to scroll both sides of the sidebar down one icon
Up to scroll both sides of the sidebar up one icon
1 to trigger the refill
2,3 and 4 to trigger the infantry buttons
5,6,7 and 8 to trigger the vehicle buttons
9 to scroll the both sides of the sidebar up one page
0 to scroll the both sides of the sidebar down one page
+ to scroll the sidebar to the end of the list
- to scroll the sidebar to the beginning of the list
A,B,C and D to change the current alternate (if the special alternate feature is enabled)

When a purchase button is clicked (or the relavent key is pressed) and the special alternate feature is disabled,
the item is purchased (just like the normal PT) and the sidebar goes away.
If the special alternate feature is enabled then the PT item corresponding to the current selected alternate is checked. If it exists, it is purchased..
Otherwise, the normal non alternate PT item is purchased.
The first button/key is for the normal non alternate PT item, the other 3 are for the 3 alternates.
If the item purchased is a refill, the refill limit is set (the refill limit is intended to prevent people who just sit there all day refilling constantly)

Whilst the sidebar is on the screen, you can still see (if they are in the default positions and havent been moved by the HUD code to somewhere the sidebar would obscure them) the radar, health bar, credits counter and time limit counter.
Also, you can see damage indicators to see that you are being attacked.
All relavent data is read from the Purchase Settings and Team Purchase Settings in leveledit.
All of the PT changing engine calls and scripts apply to the sidebar too and will be picked up by it.
Also, the sidebar is 100% compatible with the ExpVehFac scripts done by WhiteDragon
The sidebar requires VehicleBuildingDisable=true in hud.ini and also requires scripting (such as the ExpVehFac scripts)
to call Set_Currently_Building as appropriate.

There is a feature that will set some data from the client to the server (this is intended so the server can check things like if the client has a given map installed).

On the server, there is a hook that gets given the player ID of the player whos data has been recieved, the count of how many data entries exist and the data.
DO NOT store pointers to that data anywhere, if you need to keep it, make a copy.
You create a function of the form
void Data_Hook(int PlayerID,unsigned int count, unsigned int *data)
Then you pass the function to AddDataHook (defined in engine.h)
You can only have one data hook function registered at any one time. Also, if you want to have no hook at all registered, pass NULL to AddDataHook.
On the client, create the following registry keys under
HKEY_LOCAL_MACHINE\SOFTWARE\Westwood\Renegade
DataCount should be set to the count of how many data items exist
Datan should be the REG_DWORD data items where n starts at 0 and goes to DataCount-1
Both DataCount and Datan should be prefixed by the ModReg string set in hud.ini so that different mods have different data.
For example, if ModReg=RenAlert then the strings would be RenAlertDataCount and RenAlertDatan

There is a feature which changes where the code that checks for game updates reads the version from.
Set the VersionReg hud.ini keyword to a registry path (e.g. Software\Westwood\Renegade) and the Version and SKU keys will be read from there instead of the normal location).
The registry key has to go under HKEY_LOCAL_MACHINE.
Note that this only affects the game client, not the FDS at all and also these new keys will only be read for the update check, the regular keys are read for all other uses.

There is a fix that stops the problem with vehicles getting stuck near ladders (such as aircraft flying over the top of them).

There is also a hook that lets custom scripts.dll mods (e.g. server-side mods) hook whenever a player leaves the server.
You create a function of the form
void Player_Leave_Hook(int PlayerID)
Then you pass the function to AddPlayerLeaveHook (defined in engine.h)
Then, the function gets called everytime a player leaves the server
PlayerID is the player ID of the player that just left
I am unsure exactly what state things are in (if for example, you could get the player name with Get_Player_Name_By_ID).
You can only have one player leave hook function registered at any one time. Also, if you want to have no hook at all registered, pass NULL to AddPlayerLeaveHook.
This works on the server regardless of if clients have bhs.dll

There is a fix for the "wall lag", which makes sliding alongst walls go much smoother.
This feature requires bhs.dll on the client. If both the server and client have it (or have the old black-intel fix for the same thing), it will work.
There are no problems with servers who dont have it and clients who do or vice versa.

There is a change so that when C4 objects are fired, the ::Created event of any scripts on them is correctly called.

There is a change so that the windows FDS doesnt try to write into the registry "RunOnce" key anymore.

There is a change such that the HUD is not affected by Set_Screen_Fade_Color or Set_Screen_Fade_Opacity anymore.
This requires bhs.dll on the client.

There is a fix for the "turret lag" which makes turrets turn more smoothly.
This requires bhs.dll on the client.

There is a new change to the edit control and to drop down lists with editiable text. You can now use control-x to cut, control-c to copy and control-v to paste. This works with the windows clipboard and any other apps that use it.
This requires bhs.dll on the client.

There is a new dialog to configure bhs.dll features. It is accessed by pressing the "bhs.dll options" button on the options dialog and can be accessed from the main menu or in game.
On this dialog is a checkbox for the client chat log feature, a drop down list for the screenshot format feature plus a checkbox for the new high quality shadows feature.
The high quality shadows feature is part of D3D9.dll. The default is "enabled" but for cards where having it enabled causes a FPS drop (because they have horrible render target performance), you can turn it off.
For reference, the setting is saved as HighQualityShadows under the HKEY_LOCAL_MACHINE\SOFTWARE\Westwood\Renegade key, 1 = high quality shadows on, 0 = high quality shadows off.
There is a checkbox that lets you disable the custom shaders.
For reference, the setting is saved as Shaders under the HKEY_LOCAL_MACHINE\SOFTWARE\Westwood\Renegade key, 1 = shaders on, 0 = shaders off. This will take effect next time you start renegade.
There is also a list of all the keys for the custom keys feature with a way to change them. Basicly, this feature does the same job as keycfg.exe except that you cant add new keys. (you need to manually edit keys.cfg or use keycfg.exe for that)
Click on the key in the list and its current value will appear in the control. Then you can click on this control and press the new key (like the normal renegade key config dialogs)
Press "back" to exit without saving and presss "save changes" to save the changes.
Other bhs.dll features that require configuration items will go on this dialog too.
This requires bhs.dll on the client.

There is a new engine call Display_Security_Dialog. This will display the "You do not have the required security to access this terminal" dialog for a given player.

There are changes to the way renegade detects video cards so that it will detect more video cards (especially newer ones that arent detected by the old code). It will even detect all the way up to the 8800 NVIDIA card.
Also, it will correctly detect more video card driver versions. Plus, if you have a newer NVIDIA card, it will enable and use DXT1 (whereas the old renegade code would disable it).

There is a new feature, d3d9. This consists of a dll called d3d9.dll that changes renegade so it talks to Direct3D9 instead of Direct3D8.
This replaces scorpio9a's old RenD3D9 (without the fancy options rend3d9 has and without the bugs rend3d9 has). This is a required part of scripts.dll, not using d3d8.dll
(or using any other d3d8.dll such as scorpio9a's dll) will probobly lead to crashes.
This includes a feature whereby the size of the shadow texture is increased which leads to shadows that look better in game.
If you see a d3derr.log file in your renegade folder, please send it to me as it contains valuable information related to d3d9.dll.

As of 3.0, if you set VehicleOwnershipDisable to true, it will (in addition to disabling the ownership), send the CUSTOM_EVENT_VEHICLE_OWNER special custom (see scripts.h or readme.txt for details) to the vehicle object to notify it of its owner.

Shaders.dll is a new dll that implements various hooks inside the Renegade rendering pipeline replacing the fixed function code with various programmable shader classes that allow effects that have never before been possible in Renegade to come
to life. This is just the first installment and includes a few basic shaders as well as examples, with many more to come.

If shaders are disabled via the checkbox on the bhs.dll configuration dialog or if your graphics card does not support programmable shaders,
shaders.dll will be disabled and will not be called.

There is a new engine call in bhs.dll, Set_Shader_Number. See below under scene shaders to see how to use this.
There is a new engine call in bhs.dll, Set_Shader_Number_Vector. This works just like Set_Shader_Number but lets you pass a Vector4 to a shader.

To use shaders, you need a Shader Database. shaders.sdb is a global shader database. postprocess.sdb is the shader database for post
process shaders. Also, you can have a map specific shader database (so if your map was c&c_islands.mix, the shader database would be
c&c_islands.sdb).

Shader databases are created by the Shader Database editor, sdbedit.exe. To create a normal shader, select the shader type from the "Available
Shader Types" drop down, type a name (as mentioned below, this needs to match with the material that is on the mesh you want to apply this
shader to) into the "New Shader Name" box and press "New Shader". Select a shader in the "Current Shaders" box and press "Edit Shader" to
edit its settings and "Del Shader" to delete the shader. When editing a shader or scene shader, use the "open" button to open the shader
FX file. This can be a HLSL source file or a compiled shader file but it must be a proper effect with a proper technique specified.

To create a scene shader, select one from the list. Give it a name and press "new shader". Then fill in the settings. The Stacking Scene Shader allows you to "stack" scene shaders and

have them run one after the other.

Select "File-Exit" to exit sdbedit.exe. Select "File-new shader database" to create a new shader database. Select "File-open shader database" to
open an existing shader database. Select "File-save shader database" to save a shader database. Select "File-new scene shader
database" to create a new scene shader database. Select "File-open scene shader database" to open an existing scene
shader database. Select "File-save scene shader database" to save a scene shader database.

When the Render function in shaders.dll is called, all the shaders are asked if they are going to render this mesh. If none of them return TRUE
from the Can_Render function, the default shader is used (which currently goes through the fixed function pipeline but may be made into a shader
itself in the future). Inside Can_Render, the shader checks if it has been properly loaded and if the material name for the mesh being rendered
matches the name of the shader as set in sdbedit.exe. If it does, the shaders Render function is called to apply the shader.

As of 3.2, the .FX Files are no longer included into the sdb file. So you need to ship any .fx files you use along with the shaders.sdb or sceneshaders.sdb file.

NormalMappingShader (use hlslnormal_map.fx for this)

Basic single point light normal mapping shader with specular that offsets the lookup coordinates based on the eye vector from the camera
(parallax movement). Light color defines the light color to use.
Ambient color defines the ambient color to use.
Surface Color defines the surface color to use.
Specular Enabled, Specular Power and Specular Color define the specular settings.
You can use the NVIDIA's "NormalMapFilter" to generate a normal map based off of an existing texture, or you can also use NVIDIA's
"Melody" to generate a normal map based off a high polygon version of the model you are going to apply the shader to. Light Direction defines the
location of the point based light, it does not attenuate. Color is the color of the light, defined in floating point values. To get a floating point value
from an integer value, devide the integer value by 255. Note that this shader will only apply to meshes that contain vertex normal
data plus at least one set of texture coordinates.

http://developer.nvidia.com/object/photoshop_dds_plugins.html
http://developer.nvidia.com/object/melody_home.html

GlassShader (use glass.fx for this)

Reflection, refraction, and Fresnel effect shader. Uses a 6 faced cubemap texture as the environment source, must be a DDS saved as a
"Cubemap". This texture is REQUIRED. The normal texture loading code does not apply in this

shader, so you must use .dds as the extension.
Reflect Strength and Refract Strength are modifiers to the reflection and refraction strength of the shader. Included is a sample cubemap texture.

GlowShader (use glow.fx for this)

A simple multiple pass vertex directional lighted shader with glow effect. Light Direction defines the direction of the light, GlowColor is the lit glow
color, GlowAmbient is the base color of the object. These are both defined in floating point values. To get a floating point value from an integer
value, devide the integer value by 255. Thickness defines the seperation from the base color pass and the glow pass.

Included examples

Included in this package are various example textures, and two example shader databases. sceneshaders.sdb contains a bloom shader chain
implementation and can be used with no ill effects on any installation of Renegade. To use, just drop it into your data folder. shaders.sdb
contains the offset normal mapping shader applied to the GDI barracks and should be used along with the example normal map textures. This
example shader database should be used with caution as issues with the offset mapping shader lighting code does arise. These will be fixed in
scripts 3.1.
Included examples:
sceneshaders.sdb (example scene shader database)
shaders.sdb (example shader database)
bluetechno.tga (example texture for the tone map shader)
grad3.tga (example texture for the tone map shader)
heatsignature.tga (example texture for the tone map shader)
nightvis.tga (example texture for the tone map shader)
cubetexture_w3d.dds (example texture for the glass shader)
v_gdi_mammth_n.dds (example texture for the normal mapping shader)
v_nod_flame_n.dds (example texture for the normal mapping shader)

Scene Shaders
A scene shader is a shader that is applied to the entire scene after it has been rendered (but before HUD and UI is displayed on top)
The stacking scene shader lets you combine other scene shaders and have them run one after the other.
UID must be unique for the shader as it is used by the shader number send engine call to set the currently active scene shader. To set which shader is active, use the Set_Shader_Number engine call with a number of 0, and a number2 corresponding to the UID of the scene shader you wish to activate. If number is not zero,
the number and number2 will be sent to the to the current scene shader. Currently no scene shaders make use of this functionality.
Scene shaders will only load on cards supporting the correct shader version.
Also, if the scope is changed via the scope change engine calls then the current "scope shader"

for that player is set to the scene shader with the UIN matching the scope number.
When the scope is activated, the current scene shader is set to the current scope shader, when the scope is deactivated, it is restored.

Tone Map Scene Shader

A basic tone mapping shader, uses a luminance filter to get a lookup value which color value is then taken from a gradient map. Can be
used to produce nightvision, underwater, and many more effects. Texture is loaded using normal texture loading code, so be sure to have the extension as .tga in the shader editor even if the file is a dds. Use with PP_ToneMap.fx.

Bloom Scene Shader

A bloom blurring shader, takes the underlaying value and blurs it according to a series of constants. Bloom scale defines the blurring
factor. Use with PP_BloomV.fx and PP_BloomH.fx, both of which are required to produce a proper looking bloom effect.

Bright Pass Scene Shader

A bright pass shader, filters out pixels that are dimmer than the Luminance value defined in the shader editor. Use with
PP_BrightPass.fx.

Generic Scene Shader

A generic shader, takes no parameters. Use with PP_DownFilter8.fx and PP_CombineUpscale8.fx. PP_DownFilter8.fx takes the source
image and downscales it 8 times while multi-sampling the image. PP_CombineUpscale8.fx upscales the previous target by a factor of 8 before
combining it with the original scene. Use with PP_Monochrome.fx for a monochrome (i.e. greyscale) effect. Use with PP_Invert.fx for a color invert effect.

If you wish to make your own shader, the best place to start is the source code to shaders.dll to see how its done. Just study an existing shader
and copy what it does.

As of scripts.dll 3.2, most of the code responsible for drawing the custom HUD has been moved to shaders.dll.
It still works just like it did before but now if you want to, you can change shaderhud.cpp to change what it does and how it works.
Study shaderhud.cpp to see what its doing and how its drawing the HUD.

Shaders.dll exports the following functions which will be called at various times:
Release_Resources (called just before the Direct3D Device is released and reset and should be used to free any Direct3D resources allocated by
shaders.dll)

Destroy_Resources (called jsut before the Direct3D Device is going to be destoyed and should be used to free any Direct3D resources allocated by
shaders.dll)
Reload_Resources (called after the Direct3D Device is acquired and should be used to load any Direct3D resources required by the shaders)
Render (called from inside the rendering pipe, once per mesh. Is used to draw the object.
MapLoaded (called when the map is loaded)
MapUnloaded (called when the map is unloaded)
FrameStart (called when the frame drawing begins, is used to handle initialization for the post process shaders)
FrameEnd (called after the 3D has finished drawing but before any
HUD/UI/Set_Screen_Fade_xxx is drawn, is used to handle applying the post
process shaders)
ScopeTrigger (called anytime the scope is enabled or disabled)
ScopeChange (called anytime the scope is changed, the number matches with the numbers set for the scope functionality)
ShaderSet (called anytime Set_Shader_Number is called for this player)
ShaderSet2 (called anytime Set_Shader_Number_Vector is called for this player)
DrawSkin (called when skin mesh data is copied to the vertex buffer, used to add Tangent and Binormal data for normal map purposes)
DrawRigid (called when rigid non skin mesh data is copied to the vertex buffer, used to add Tangent and Binormal data for normal map purposes)
Cleanup_HUD2 (called to cleanup the hud stuff in shaderhud.cpp)
ReadHUDBits (called to read the relavent bits from hud.ini that shaderhud.cpp needs)
ReadRuntimeINI (called to read the hud.ini runtime overrides)
UpdateHUD2 (called when the HUD is to be drawn)
Update_Radar_Map (called when the radar map details for the custom radar map need updating)
Send_HUD_Number (called when the Send_HUD_Number engine call is used)
ScreenFadeRender (called when the screen fade overlay as set by Set_Screen_Fade_Opacity and Set_Screen_Fade_Color is to be drawn)
SetScriptNotify (used to pass to shaders.dll the ScriptNotify function used in sending data back to scripts.dll)
These next ones are used so that certain pieces of code will pass through shaders.dll and through the shader state manager and other stuff
If you dont know what you are doing, leave them alone.
SetRenderState
SetTextureStageState
DynamicVBReset
SetViewport
SetRenderTarget
SetTransform
DoFont
CopySurface

There is a new function, vsync. It is enabled and disabled through the bhs.dll configuration dialog and defaults to on.
When it is enabled, renegade will syncronise draw calls with the refersh rate of your monitor.
For reference, the setting is saved as VSync under the

HKEY_LOCAL_MACHINE\SOFTWARE\Westwood\Renegade key. 1 = on, 0 = off.

There is new code that will aid in finding memory leaks in scripts.dll/shaders.dll/etc. Basicly you build a "debug" build of the dll and run it in the game.
It will spit out some log files containing details of anything it thinks is a memory leak along with the filename and line number of the source file that caused the leak.
Other log files will be created with errors for such things as allocating memory with new[] and freeing it with delete instead of delete[]

There is a new patch to fix a bug whereby the game may leak ScriptImpClass objects when shutdown.
There are also a whole bunch of memory leak fixes to all binaries as a result of the memory leak detection code.

There is now support for multisample anti-aliasing. Select the option you want on the bhs.dll config dialog and it will be used when you restart renegade.
For reference, the setting is saved as AA under the HKEY_LOCAL_MACHINE\SOFTWARE\Westwood\Renegade key. the number indicates what level of anti-alias you want.

There is a new engine call SendHUDNumber which is used to send a number to a given player that the custom HUD code in shaders.dll can pick up and use. See the shaders.dll section for details.

There is a new engine call GetExplosionObj. If you are in a ::Killed event and the object that did the killing was a C4 or Beacon object, this engine call will retieve that object for you to interrogate.
Note that you cannot hang on to pointers to the returned object or anything it may point to (names etc)
If you wish to preserve the info, you need to duplicate it yourself in your own code.
As of 3.2.2, I fixed this to work for buildings too (you will now get the correct object when inside the ::Killed event of a script attached to a building controler)

There is a new feature which lets you override the registry key where the game reads the WOL URLs (news etc)
Set WOLUrlReg in hud.ini to the name of a registry key under HKEY_LOCAL_MACHINE\SOFTWARE\Westwood\Renegade (default is WOLSettings\URL) and the code will read the URLs from there instead of from the normal location.
HTML to be loaded via these URLs needs <!-- Certified Westwood Studios --> as the first line of the file.

Also, the values of the AA, vsync, shaders and shadow settings are now output to sysinfo.txt

The code for texture based fonts (such as font6x8.tga) has been rewritten to be faster by removing the use of the CopyRects Direct3d function.
The remaining places that use CopyRects have also been changed to not use it anymore.
There is also a change to make the DIBSection used by the TTF file based font code larger.
This means that really large (width or height) characters wont be cut off anymore.

There is a feature in shaders.dll that lets you load a plugin.
See shaderplugin_example for an example of an empty plugin you can start with as a base.
To load the plugin add something like this in hud.ini
[ShaderPlugins]
PluginCount=1
PluginsRequired=true
1=plugin.dll

Added a new vehicle management dialog to bhs.dll. Set the Vehicle logical key in keys.cfg to a value.
Then, if you are the driver of a vehicle, you can press that key.
Up comes a dialog showing the contents of the vehicle.
You can click on a player name to see what preset they are. Also, you can then kick them out of the vehicle
(anyone other than the driver can be kicked out) or set anyone other than the current gunner or driver to be the gunner.

There is a feature that allows a shader to send a notification back to a script.
On the shaders.dll side, call Notify(ID,notify) where ID is the ID that was registered by the script and notify is the number to send.
On the scripts.dll side, create a ShaderNotifyStruct structure.
Set ID to the ID you want to listen for. Set Player_ID to the player ID you want to listen for. If Player_ID is -1, it will match on any player ID.
Set data to some data that will be passed back into your hook function
And set hook to your hook function.
The hook function looks something like this
void ShaderNotify(void *data, int notify)
data is what you set in the ShaderNotifyStruct structure before
notify is what was sent via the call to Notify in shaders.dll
Then you call AddShaderNotify. Call RemoveShaderNotify and pass it the value returned from AddShaderNotify to remove the notification.

There is a new feature in bhs.dll that changes the default sort order of the WOL Server list dialog to sort by current player count instead of by name as a default