
Subject: Scripts.dll problem

Posted by [hatstand](#) on Wed, 16 Jan 2008 12:03:37 GMT

[View Forum Message](#) <> [Reply to Message](#)

first off, yes i HAVE followed ALL of the instructions in the stickied topic.

ok. ive run the code itself past someone who knows what theyre doing, it all checks out (im using the ssgm dll source), im still getting these two errors:

Error 2 error LNK2019: unresolved external symbol "int __cdecl __CxxSetUnhandledExceptionHandler(void)" (?__CxxSetUnhandledExceptionHandler@@@YAHXZ) referenced in function _NotifyHookUnhandledExceptionHandler dllmain.obj

and

Error 3 fatal error LNK1120: 1 unresolved externals scripts.dll

edit: the code:

Toggle Spoiler/* Renegade Scripts.dll

SSGM main functions and classes

Copyright 2007 Vloktboky, Whitedragon(MDB), Mac, Jonathan Wilson

This file is part of the Renegade scripts.dll

The Renegade scripts.dll is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version. See the file COPYING for more details.

In addition, an exemption is given to allow Run Time Dynamic Linking of this code with any closed source module that does not contain code covered by this licence.

Only the source code to the module(s) containing the licenced code has to be released.

*/

```
#include <fstream>
#include <stdarg.h>
#include "scripts.h"
#include "date.h"
#include "engine.h"
#include "gmmain.h"
#ifdef WIN32
#include <ddeml.h>
#else
#include <dlfcn.h>
#endif
```

```
bool OkLoad = false;
```

```

#ifndef WIN32
extern void *bhs;
UnloaderClass Unloader;
#endif
DataStruct *Data = 0;
SettingsStruct *Settings = 0;

//*****
//***** SETTINGS STUFF *****
//*****

void SSGMSettings::Load() {
    LoadSString(FDSLogRoot,"FDSLogRoot", "ssgm",true,false);
    LoadBool(Gamelog,"EnableGamelog",true,true,false);
    LoadBool(NewGamelog,"EnableNewGamelog",true,true,false);
    LoadBool(EnableLogging,"EnableLog",true,true,false);
    LoadBool(EchoLog,"EchoLog",true,true,false);
    LoadBool(MergeGamelogRenlog,"WriteGamelogtoSSGMlog",false,true,false);
    if (MergeGamelogRenlog) {
        Gamelog_Archive_Logfiles = false;
    }
    else {
        LoadBool(Gamelog_Archive_Logfiles,"GamelogArchiveLogfiles",false,true,false,false);
    }

    LoadInt(GameMode,"GameMode");
    if (GameMode > 5 || GameMode < 1) {
        GameMode = 1;
        FDSMessage("Gamemode out of range. Defaulting to AOW.", "_ERROR");
    }

    //Now isn't this just a fun 'little' piece of code?
    if (GameMode == 3 || GameMode == 4) {
        SniperChars.clear();
        std::vector<std::string> Snipers;
        LoadList(Snipers,"Snipers",false);
        bool Found[2] = {false,false};
        if (!Snipers.empty()) {
            unsigned int FirstSniperID[2] = {0,0};
            unsigned int FirstSniperStringID[2] = {0,0};
            std::string FirstSniperIcon[2];

            //This loop finds the first sniper character for each team.
            for (unsigned int Team = 0; Team < 2; Team++) {
                unsigned int j = 0;
                unsigned int i = 0;

```

```

PurchaseSettingsDefClass *p = Get_Purchase_Definition(CHARACTERS,PTTEAM(Team));
TeamPurchaseSettingsDefClass *t = Get_Team_Purchase_Definition(PTTEAM(Team));

for (j = 0; j < 4; j++) {
    const char *Name = Get_Definition_Name(t->presetids[j]);
    if (InVector(Snipers,Name)) {
        FirstSniperID[Team] = t->presetids[j];
        FirstSniperStringID[Team] = t->stringids[j];
        FirstSniperIcon[Team] = t->textures[j].Peek_Buffer();
        Found[Team] = true;
        if (GameMode == 4) {
            SniperChars.push_back(Name);
        }
        break;
    }
}

if (!Found[Team]) {
    for (j = 0; j < 10; j++) {
        if (p->presetids[j]) {
            const char *Name = Get_Definition_Name(p->presetids[j]);
            if (InVector(Snipers,Name)) {
                FirstSniperID[Team] = p->presetids[j];
                FirstSniperStringID[Team] = p->stringids[j];
                FirstSniperIcon[Team] = p->textures[j].Peek_Buffer();
                Found[Team] = true;
                if (GameMode == 4) {
                    SniperChars.push_back(Name);
                }
                break;
            }
        }
        for (i = 0; i < 3; i++) {
            if (p->altpresetids[j][i]) {
                Name = Get_Definition_Name(p->altpresetids[j][i]);
                if (InVector(Snipers,Name)) {
                    FirstSniperID[Team] = p->altpresetids[j][i];
                    FirstSniperStringID[Team] = p->stringids[j];
                    FirstSniperIcon[Team] = p->alttextures[j][i].Peek_Buffer();
                    Found[Team] = true;
                    if (GameMode == 4) {
                        SniperChars.push_back(Name);
                    }
                }
                j = 10; //Break outer loop
                break;
            }
        }
    }
}
}
}
}
}

```

```

}
}
}

if (!Found[0] && !Found[1]) {
    GameMode = 1;
    FDSMessage("No sniper characters found. Defaulting to AOW.", "_ERROR");
}
else {

    Change_Spawn_Char(0,Get_Definition_Name(FirstSniperID[0]));
    Change_Spawn_Char(1,Get_Definition_Name(FirstSniperID[1]));

    //This loop replaces any non sniper characters with the first sniper character found.
    for (unsigned int Team = 0; Team < 2; Team++) {
        unsigned int j = 0;
        unsigned int i = 0;
        PurchaseSettingsDefClass *p = Get_Purchase_Definition(CHARACTERS,PTTEAM(Team));
        TeamPurchaseSettingsDefClass *t = Get_Team_Purchase_Definition(PTTEAM(Team));

        if (Found[Team]) {
            for (j = 0; j < 10; j++) {
                p->costs[j] = 0;
                if (GameMode == 4) {
                    p->presetids[j] = FirstSniperID[Team];
                    p->stringids[j] = FirstSniperStringID[Team];
                    p->textures[j] = FirstSniperIcon[Team].c_str();
                    p->altpresetids[j][0] = 0;
                    p->altpresetids[j][1] = 0;
                    p->altpresetids[j][2] = 0;
                }
                else if (p->presetids[j]) {
                    bool IsSniper = false;
                    const char *Name = Get_Definition_Name(p->presetids[j]);
                    if (InVector(Snipers,Name)) {
                        SniperChars.push_back(Name);
                        IsSniper = true;
                    }
                    else {
                        p->presetids[j] = FirstSniperID[Team];
                        p->stringids[j] = FirstSniperStringID[Team];
                        p->textures[j] = FirstSniperIcon[Team].c_str();
                    }
                }
                for (i = 0; i < 3; i++) {
                    if (p->altpresetids[j][i]) {
                        Name = Get_Definition_Name(p->altpresetids[j][i]);
                        if (InVector(Snipers,Name)) {
                            SniperChars.push_back(Name);
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    else if (IsSniper) {
        p->altpresetids[j][i] = p->presetids[j];
        p->alttextures[j][i] = p->textures[j];
    }
    else {
        p->altpresetids[j][i] = FirstSniperID[Team];
        p->alttextures[j][i] = FirstSniperIcon[Team].c_str();
    }
}
}
}
}
for (unsigned int j2 = 0; j2 < 4; j2++) {
    if (GameMode == 4) {
        t->presetids[j2] = FirstSniperID[Team];
        t->stringids[j2] = FirstSniperStringID[Team];
        t->textures[j2] = FirstSniperIcon[Team].c_str();
    }
    else if (t->presetids[j2]) {
        const char *Name = Get_Definition_Name(t->presetids[j2]);
        if (InVector(Snipers,Name)) {
            SniperChars.push_back(Name);
        }
        else {
            t->presetids[j2] = FirstSniperID[Team];
            t->stringids[j2] = FirstSniperStringID[Team];
            t->textures[j2] = FirstSniperIcon[Team].c_str();
        }
    }
}
}
}
}
//Update_PT_Data();
}
}
else {
    GameMode = 1;
    FDSMessage("No sniper characters found. Defaulting to AOW.", "_ERROR");
}
}
}

```

```

char Mode[50];
if (GameMode == 1) sprintf(Mode,"All Out War");
else if (GameMode == 2) sprintf(Mode,"Capture The Flag");
else if (GameMode == 3) sprintf(Mode,"Snipers Only");
else if (GameMode == 4) sprintf(Mode,"500 Snipers Only");

```

```

else if (GameMode == 5) sprintf(Mode,"Infantry Only");
FDSMessage(StrFormat("Running in %s mode.",Mode),"_GENERAL");

LoadBool(LogBuildingDamage,"LogBuildingUnderAttack");
LoadBool(LogBuildingKills,"LogBuildingKills");
LoadBool(BuildingDeathRewardPage,"EnableBuildingDeathRewardPage",false);
LoadFloat(BuildingDamageInt,"BuildingDamageInterval",30.0);

LoadBool(AFKKick,"EnableAFKKick",false);
if (AFKKick) {
    LoadInt(AFKWait,"AFKWait",10);
    if (AFKWait < 2){
        FDSMessage("AFKWait must be at least 2 minutes. Defaulting to 10 minutes.", "_ERROR");
        AFKWait = 10;
    }
    LoadSString(AFKPageMessage,"AFKPageMessage","Warning! You might be kicked for being
AFK/Idle if you do not move!");
}

LoadBool(DropWeapons,"EnableDropWeapons");
LoadBool(LogPlayerPurchase,"LogCharacterPurchases");
LoadBool(LogPlayerKills,"LogPlayerKills");
LoadBool(LogVehiclePurchase,"LogVehiclePurchases");
LoadBool(Weather,"EnableWeather",false);
LoadBool(LogVehicleKills,"LogVehicleKills");
LoadBool(EnableVehicleDamageAnim,"EnableVehicleDamageAnimations");
LoadBool(EnableVehicleDeathAnim,"EnableVehicleDeathAnimations");
LoadBool(DestroyPlayerVeh,"DestroyPlayerVeh",false);
LoadBool(EnableVehicleWreckages,"EnableVehicleWreckages");
LoadBool(DefenseShell,"DefenseShootWrecks",false);

LoadBool(OBGEnable,"EnableObGlitchProtect",false);
if (OBGEnable) {
    LoadSString(OBGPageMessage,"ObGlitchPageMessage","Warning! You might be kicked for
glitching the Obelisk!");
}

LoadList(SvSKillMsg,"SvSKillMsg");
LoadList(VvSKillMsg,"VvSKillMsg");

std::string DisableListName;
LoadSString(DisableListName,"DisableList","ERROR",true,true,false);
DisableList.clear();
if (DisableListName != "ERROR") {
    for (int z = 1; ; ++z) {
        char value[500];
        INI->Get_String(DisableListName.c_str(),StrFormat("%02d",z).c_str(),"NULL",value,500);
        if (!strcmp(value,"NULL")) break;
    }
}

```

```

DisableList.push_back(value);
Disable_Enlisted_By_Name(0,value);
Disable_Enlisted_By_Name(1,value);
Disable_Preset_By_Name(0,value);
Disable_Preset_By_Name(1,value);
}
}

LoadBool(DisableBaseDefenses,"DisableBaseDefenses",false);
LoadBool(DisablePowerPlants,"DisablePowerPlants",false);
LoadBool(DisableRefineries,"DisableRefineries",false);
LoadBool(DisableSoldierFactories,"DisableSoldierFactories",false);
LoadBool(DisableVehicleFactories,"DisableVehicleFactories",false);
LoadBool(DisableRepairPads,"DisableRepairPads",false);
LoadBool(DisableCommCenters,"DisableCommCenters",false);

LoadBool(SpawnWeap,"EnableSpawnWeapons");
The_Cnc_Game()->SpawnWeapons = SpawnWeap;

std::string WSL1,WSL2,WSO;

LoadSString(WSL1,"WeaponStartEngL1","DefaultEngL1",true,true,false);
WeaponStartEngL1.clear();
for (int z = 1; ; ++z) {
    char value[500];
    INI->Get_String(WSL1.c_str(),StrFormat("%02d",z).c_str(),"NULL",value,500);
    if (!strcmp(value,"NULL")) break;
    if (GameMode == 2) {
        AmmoDefinitionClass *Def =
Get_Weapon_Ammo_Definition(Get_Powerup_Weapon(value),true);
        if (Def) {
            int Mode2 = Def->AmmoType.Get();
            if (Mode2 == 0 || Mode2 == 2) {
                WeaponStartEngL1.push_back(value);
            }
        }
    }
    else if (GameMode == 3 || GameMode == 4) {
        AmmoDefinitionClass *Def =
Get_Weapon_Ammo_Definition(Get_Powerup_Weapon(value),true);
        if (Def) {
            int Mode2 = Def->AmmoType.Get();
            if (Mode2 == 0) {
                WeaponStartEngL1.push_back(value);
            }
        }
    }
    else {

```

```

WeaponStartEngL1.push_back(value);
}
}
if (WeaponStartEngL1.empty() && (GameMode == 1 || GameMode == 5)) {
WeaponStartEngL1.push_back("CnC_POW_MineTimed_Player_01");
WeaponStartEngL1.push_back("POW_Pistol_Player");
WeaponStartEngL1.push_back("CnC_POW_MineRemote_02");
}

LoadSString(WSL2,"WeaponStartEngL2","DefaultEngL2",true,true,false);
WeaponStartEngL2.clear();
for (int z = 1; ; ++z) {
char value[500];
INI->Get_String(WSL2.c_str(),StrFormat("%02d",z).c_str(),"NULL",value,500);
if (!strcmp(value,"NULL")) break;
if (GameMode == 2) {
AmmoDefinitionClass *Def =
Get_Weapon_Ammo_Definition(Get_Powerup_Weapon(value),true);
if (Def) {
int Mode2 = Def->AmmoType.Get();
if (Mode2 == 0 || Mode2 == 2) {
WeaponStartEngL2.push_back(value);
}
}
}
else if (GameMode == 3 || GameMode == 4) {
AmmoDefinitionClass *Def =
Get_Weapon_Ammo_Definition(Get_Powerup_Weapon(value),true);
if (Def) {
int Mode2 = Def->AmmoType.Get();
if (Mode2 == 0) {
WeaponStartEngL2.push_back(value);
}
}
}
else {
WeaponStartEngL2.push_back(value);
}
}
if (WeaponStartEngL2.empty() && (GameMode == 1 || GameMode == 5)) {
WeaponStartEngL2.push_back("CnC_POW_MineRemote_02");
WeaponStartEngL2.push_back("POW_Pistol_Player");
WeaponStartEngL2.push_back("CnC_POW_MineTimed_Player_02");
WeaponStartEngL2.push_back("CnC_MineProximity_05");
}

LoadSString(WSO,"WeaponStartOther","DefaultOther",true,true,false);
WeaponStartOther.clear();

```



```

for (int z = 1; ; ++z) {
    char value[500];
    INI->Get_String(WSO.c_str(),StrFormat("%02d",z).c_str(),"NULL",value,500);
    if (!strcmp(value,"NULL")) break;
    if (GameMode == 2) {
        AmmoDefinitionClass *Def =
Get_Weapon_Ammo_Definition(Get_Powerup_Weapon(value),true);
        if (Def) {
            int Mode2 = Def->AmmoType.Get();
            if (Mode2 == 0 || Mode2 == 2) {
                WeaponStartOther.push_back(value);
            }
        }
    }
    else if (GameMode == 3 || GameMode == 4) {
        AmmoDefinitionClass *Def =
Get_Weapon_Ammo_Definition(Get_Powerup_Weapon(value),true);
        if (Def) {
            int Mode2 = Def->AmmoType.Get();
            if (Mode2 == 0) {
                WeaponStartOther.push_back(value);
            }
        }
    }
    else {
        WeaponStartOther.push_back(value);
    }
}
if (WeaponStartOther.empty() && (GameMode == 1 || GameMode == 5)) {
    WeaponStartOther.push_back("CnC_POW_MineTimed_Player_01");
    WeaponStartOther.push_back("POW_Pistol_Player");
}

LoadBool(CombatRefill,"BlockCombatRefills",false);

LoadBool(DisableBeacons,"DisableBeacons",false);
if (GameMode == 3 || GameMode == 4 || DisableBeacons) {
    Set_Beacon(0,0,0,0,"");
    Set_Beacon(1,0,0,0,"");
    //Update_PT_Data();
}

LoadBool(InvinBuild,"InvincibleBuildings",false);
LoadBool(EnableVehOwn,"EnableVehicleOwnership");

if (GameMode != 3 && GameMode != 4) {
    std::string Char;
    LoadSString(Char,"SpawnChar0","CnC_Nod_MiniGunner_0",true,true,false);
}

```

```

if (Char != Get_Spawn_Char(0)) {
    if (!ShowSetChangeMess) {
        ShowSetChangeMess = true;
        FDSMessage("New settings detected and loaded!","_GENERAL");
    }
    Change_Spawn_Char(0,Char.c_str());
}
LoadSString(Char,"SpawnChar1","CnC_GDI_MiniGunner_0",true,true,false);
if (Char != Get_Spawn_Char(1)) {
    if (!ShowSetChangeMess) {
        ShowSetChangeMess = true;
        FDSMessage("New settings detected and loaded!","_GENERAL");
    }
    Change_Spawn_Char(1,Char.c_str());
}
}
}

```

```

LoadInt(ForceTeam,"ForceTeam",-1);
LoadBool(BWDDetector,"EnableBWDDetector");

```

```

LoadBool(InfiniteAmmo,"EnableInfiniteAmmo",false);
if (InfiniteAmmo) {
    Enable_Infinite_Ammo();
}

```

```

LoadBool(ExtraKillMessages,"EnableExtraKillMessages");

```

```

GameObject *Temp = Commands->Create_Object("Invisible_Object",Vector3(0.0f,0.0f,0.0f));
if (Weather) {
    LoadSString(WeatherType,"WeatherType","Rain");
    Commands->Attach_Script(Temp,"MDB_SSGM_Random_Weather","");
}
if (BWDDetector) {
    LoadInt(BWDDefault,"BWDDefault",56000);
    LoadBool(BWDPunishKick,"KickBWDExploiters");
    Commands->Attach_Script(Temp,"MDB_SSGM_Manager","");
}

```

```

LoadBool(LogPowerupPurchase,"LogPowerupPurchases");
LoadBool(LogBeacons,"LogBeacons");
LoadBool(LogC4,"LogC4");
}

```

```

bool SSGMSettings::Is_Disabled(const char *Preset) {
    if (!DisableList.empty()) {
        std::vector<std::string>::const_iterator it;
        for (it = DisableList.begin();it != DisableList.end(); ++it) {

```

```

    if ((*it) == Preset) {
        return true;
    }
}
return false;
}

bool SSGMSettings::Is_Disabled(GameObject *obj) {
    return Is_Disabled(Commands->Get_Preset_Name(obj));
}

bool SSGMSettings::Is_Sniper(const char *Preset) {
    if (!SniperChars.empty()) {
        std::vector<std::string>::const_iterator it;
        for (it = SniperChars.begin(); it != SniperChars.end(); ++it) {
            if ((*it) == Preset) {
                return true;
            }
        }
    }
    return false;
}

bool SSGMSettings::Is_Sniper(GameObject *obj) {
    return Is_Sniper(Commands->Get_Preset_Name(obj));
}

void SettingsStruct::Load() {
    SettingsLoader::Load();
    SSGMSettings::Load();
    CrateSettings::Load();
    CTFSettings::Load();
}

//*****
//***** DATA STRUCT STUFF *****
//*****

#ifdef WIN32
bool PluginInfo::Load(HMODULE PHandle, const char *FileName) {
#else
bool PluginInfo::Load(void *PHandle, const char *FileName) {
#endif
    Handle = PHandle;
    File = FileName;
    Name = FileName;
    Version = "1.0";
}

```

```
Type = StandAlone;
SSGMVerRequired = false;
```

```
ChatHookHandle = 0;
HostHookHandle = 0;
PlayerJoinHookHandle = 0;
PlayerLeaveHookHandle = 0;
LevelLoadedHookHandle = 0;
GameOverHookHandle = 0;
ConsoleOutputHookHandle = 0;
DDEHookHandle = 0;
```

```
PluginInit PInit = (PluginInit)Get_Address("SSGM_Plugin_Load");
PluginCheckVersion PVersion = (PluginCheckVersion)Get_Address("SSGM_Check_Version");
CreateScriptHandle = (cs)Get_Address("Create_Script");
if (PInit && PVersion) {
    bool AllowLoad = PVersion(SSGMVersion,(PluginInfo*)this);
    if (!AllowLoad) {
        Console_Output("Failed to load plugin %s: The plugin has reported an error and stopped the
loading process.\n",Name.c_str());
        return false;
    }
    if (SSGMVersion != SSGMVer) {
        if (SSGMVerRequired) {
            Console_Output("Failed to load plugin %s: This plugin requires SSGM v%s. You are running
v%s.\n",Name.c_str(),SSGMVer.c_str(),SSGMVersion);
            return false;
        }
        else {
            Console_Output("Warning: Plugin %s was designed to work with SSGM v%s. You may
experience instability running with v%s.\n",Name.c_str(),SSGMVer.c_str(),SSGMVersion);
        }
    }
}
```

```
#ifdef WIN32
```

```
    PInit(GetModuleHandle("bhs.dll"),Settings,Data,SSGMVersion,(PluginInfo*)this);
```

```
#else
```

```
    PInit(bhs,Settings,Data,SSGMVersion,(PluginInfo*)this);
```

```
#endif
```

```
ChatHookHandle = (ChatHook)Get_Address("SSGM_Chat_Hook");
HostHookHandle = (HostHook)Get_Address("SSGM_Host_Hook");
PlayerJoinHookHandle = (PlayerJoin)Get_Address("SSGM_Player_Join_Hook");
PlayerLeaveHookHandle = (PlayerLeave)Get_Address("SSGM_Player_Leave_Hook");
LevelLoadedHookHandle = (LoadLevelHook)Get_Address("SSGM_Level_Loaded_Hook");
GameOverHookHandle = (LoadLevelHook)Get_Address("SSGM_GameOver_Hook");
ConsoleOutputHookHandle =
(ConsoleOutputHook)Get_Address("SSGM_Console_Output_Hook");
DDEHookHandle = (DDEHook)Get_Address("SSGM_DDE_Hook");
```

```

    Console_Output("Plugin %s v%s loaded\n",Name.c_str(),Version.c_str());
    Type = Plugin;
}
#ifdef WIN32
else if (CreateScriptHandle) {
    Console_Output("Scripts.dll file %s loaded\n",FileName);
    Type = Scripts;
}
#endif
else {
    Type = StandAlone;
#ifdef WIN32
    Console_Output("Stand alone .DLL %s loaded\n",FileName);
#else
    Console_Output("Stand alone .SO %s loaded\n",FileName);
#endif
}
return true;
}

void PluginInfo::Unload() {
    PluginUnload UL = (PluginUnload)Get_Address("SSGM_Plugin_Unload");
    if (UL) {
        UL();
    }
#ifdef WIN32
    FreeLibrary(Handle);
#else
    dlclose(Handle);
#endif
}

void *PluginInfo::Get_Address(const char *Func) {
#ifdef WIN32
    return GetProcAddress(Handle,Func);
#else
    return dlsym(Handle,Func);
#endif
}

SSGMData::SSGMData() {
    Mod = 0;
    PlayerPurchaseHookID = 0;
    VehiclePurchaseHookID = 0;
    ObjectHookID = 0;
    ObjectHookStruct = 0;
    AllowGamelogWrite = false;
}

```

```

std::string &SSGMDData::Get_Mod_Name() {
    if (Mod >= Settings->ModNames.size()) {
        return Settings->ModNames[1];
    }
    return Settings->ModNames[Mod];
}

void SSGMDData::Add_Chat_Command(ChatCommandClass *Ptr,const char *Command,int
ChatType,int NumParams,int GameMode) {
    ChatCommandInfo *Temp = new ChatCommandInfo;
    Temp->Ptr = Ptr;
    Temp->Command = Command;
    Temp->ChatType = ChatType;
    Temp->NumParams = NumParams;
    Temp->GameMode = GameMode;
    Ptr->Info = Temp;
    this->Commands.push_back(Temp);
}

void SSGMDData::Copy_Chat_Commands(const std::vector<ChatCommandInfo*> *List) {
    if (!List->empty()) {
        std::vector<ChatCommandInfo*>::const_iterator it;
        for (it = List->begin();it != List->end(); ++it) {
            this->Commands.push_back(*it);
        }
    }
}

void SSGMDData::Trigger_Chat_Command(int ID,int Type,const std::string &Command,const
TokenClass &Text) {
    if (!this->Commands.empty()) {
        std::vector<ChatCommandInfo*>::const_iterator it;
        for (it = this->Commands.begin(); it != this->Commands.end(); ++it) {
            if ((*it)->Command == Command && ((*it)->GameMode == Settings->GameMode ||
(*it)->GameMode == 0) && ((*it)->ChatType == Type || (*it)->ChatType == 2)) {
                if (Text.size() >= (*it)->NumParams) {
                    (*it)->Ptr->Triggered(ID,Text,Type);
                }
            }
            else {
                (*it)->Ptr->Error(ID,1,Text.size());
            }
        }
    }
}

```

```

//*****
//***** HOOKS *****
//*****

```

```

void Chat(int ID, TextMessageEnum Type, const wchar_t *Msg2) {
if (!Data->Plugins.empty()) {
std::vector<PluginInfo*>::const_iterator it;
for (it = Data->Plugins.begin();it != Data->Plugins.end(); ++it) {
if ((*it)->Type == Plugin) {
if ((*it)->ChatHookHandle) {
(*it)->ChatHookHandle(ID,Type,Msg2);
}
}
}
}
}

```

```

if (Type == 2) {
return;
}

```

```

if (Type == 0 || Type == 1){
const char *Msg3 = WideCharToChar(Msg2);
char name[50];
sprintf(name,Get_Player_Name_By_ID(ID));
GameObject *obj = Get_GameObj(ID);
if (strncmp(Msg3,"!mutate",7) == 0) {
if (obj) {
Change_Character(obj,"CnC_Nod_Mutant_0_Mutant");
Set_Max_Health(obj, 200);
Set_Max_Shield_Strength(obj, 0);
}
}
}

```

```

}

if (Type == 0 || Type == 1){
const char *Msg3 = WideCharToChar(Msg2);
char name[50];
sprintf(name,Get_Player_Name_By_ID(ID));
GameObject *obj = Get_GameObj(ID);
if (strncmp(Msg3,"!tank",5) == 0) {
if (obj) {
Change_Character(obj,"CnC_Nod_Mutant_2SF_Templar");
Set_Max_Health(obj, 1);
Set_Max_Shield_Strength(obj, 300);
Commands->Set_Shield_Type(obj, "CNCVehicleMedium");
}
}
}

```

```

}

if (Type == 0 || Type == 1){
    const char *Msg3 = WideCharToChar(Msg2);
    char name[50];
    sprintf(name,Get_Player_Name_By_ID(ID));
    GameObject *obj = Get_GameObj(ID);
    if (strncmp(Msg3,"!mp",3) == 0) {
        if (obj) {
            Change_Character(obj,"GDI_MP");
            Set_Max_Health(obj, 100);
            Set_Max_Shield_Strength(obj, 100);
        }
    }
}

}

if (Type == 0 || Type == 1){
    const char *Msg3 = WideCharToChar(Msg2);
    char name[50];
    sprintf(name,Get_Player_Name_By_ID(ID));
    GameObject *obj = Get_GameObj(ID);
    if (strncmp(Msg3,"!commando",9) == 0) {
        if (obj) {
            Change_Character(obj,"GDI_Logan_Sheppard");
            Set_Max_Health(obj, 150);
            Set_Max_Shield_Strength(obj, 50);
        }
    }
}

}

if (Settings->Gamelog && Settings->NewGamelog) {
    Gamelog_Chat_Hook(ID,Type,Msg2);
}

std::string Msg = WideCharToString(Msg2);
if (Msg[0] == '!' && !Data->Commands.empty()) {
    TokenClass Text(Msg);
    std::string Command = Text[1];
    Text.erase(1);
    Data->Trigger_Chat_Command(ID,Type,Command,Text);
}
}

void HostChat(int ID, TextMessageEnum Type, const char *Msg) {
    if (!Data->Plugins.empty()) {

```



```

std::vector<PluginInfo*>::const_iterator it;
for (it = Data->Plugins.begin();it != Data->Plugins.end(); ++it) {
    if ((*it)->Type == Plugin) {
        if ((*it)->HostHookHandle) {
            (*it)->HostHookHandle(ID,Type,Msg);
        }
    }
}
}
}
}
}

```

```

void Player_Join_Hook(int i,const char *Nick) {
    if (!Data->Plugins.empty()) {
        std::vector<PluginInfo*>::const_iterator it;
        for (it = Data->Plugins.begin();it != Data->Plugins.end(); ++it) {
            if ((*it)->Type == Plugin) {
                if ((*it)->PlayerJoinHookHandle) {
                    (*it)->PlayerJoinHookHandle(i,Nick);
                }
            }
        }
    }
    if (Settings->GameMode == 2) {
        CTF_Player_Join(i);
    }
}

```

```

void Player_Leave_Hook(int ID) {
    if (!Data->Plugins.empty()) {
        std::vector<PluginInfo*>::const_iterator it;
        for (it = Data->Plugins.begin();it != Data->Plugins.end(); ++it) {
            if ((*it)->Type == Plugin) {
                if ((*it)->PlayerLeaveHookHandle) {
                    (*it)->PlayerLeaveHookHandle(ID);
                }
            }
        }
    }
}
}
}
}

```

```

void ObjectHookCall(void *data,GameObject *obj) {
    if (Settings->Is_Disabled(obj)) {
        Commands->Destroy_Object(obj);
        return;
    }
    else if (Commands->Is_A_Star(obj)) {
        if ((Settings->GameMode == 3 || Settings->GameMode == 4) && !Settings->Is_Sniper(obj)) {
            Commands->Destroy_Object(obj);
        }
    }
}

```

```

return;
}
else {
    Attach_Script_Once(obj,"MDB_SSGM_Player","");
}
}
else if (Is_Vehicle(obj)) {
    if (Is_Harvester_Preset(obj)) {
        Attach_Script_Once(obj,"MDB_SSGM_Harvester","");
    }
    else if (Get_Vehicle_Mode(obj) == TURRET) {
        Attach_Script_Once(obj,"MDB_SSGM_Base_Defense","");
    }
    else if (!isin(Commands->Get_Preset_Name(obj),"Destroyed")) {
        Attach_Script_Once(obj,"MDB_SSGM_Vehicle","");
    }
}
}
else if (Is_Beacon(obj)) {
    if (Settings->GameMode == 2 || Settings->GameMode == 3 || Settings->GameMode == 4 ||
Settings->DisableBeacons) {
        Commands->Destroy_Object(obj);
    }
    else if (Settings->LogBeacons) {
        Attach_Script_Once(obj,"MDB_SSGM_Beacon","");
    }
}
}
else if (Is_Building(obj)) {
    Attach_Script_Once(obj,"MDB_SSGM_Building","");
}
}
else if (Is_Powerup(obj)) {
    if (!Data->Mod && isin(Commands->Get_Preset_Name(obj),"Crate")) {
        Attach_Script_Once(obj,"MDB_SSGM_Crate","");
    }
    if ((Settings->GameMode == 3 || Settings->GameMode == 4) &&
_stricmp(Get_Powerup_Weapon_By_Obj(obj),"None")) {
        Commands->Destroy_Object(obj);
        return;
    }
}
}
}
else if (Is_C4(obj)) {
    Attach_Script_Once(obj,"MDB_SSGM_C4","");
}
}
else if (Is_Cinematic(obj)) {
    if (strstr(Commands->Get_Preset_Name(obj),"Beacon_Ion_Cannon_Anim_Pre")) {
        FDSMessage("Ion Cannon Strike initiated", "_BEACON");
    }
}
else if (strstr(Commands->Get_Preset_Name(obj),"Beacon_Nuke_Strike_Anim_Pre")) {

```

```

    FDSMessage("Nuclear Strike initiated", "_BEACON");
}
}
if (Settings->Gamelog) {
    Gamelog_ObjectHook(obj);
}
if (Settings->GameMode == 2) {
    CTF_ObjectHook(obj);
}
}
}

```

```

void SSGM_Purchase_Hook(BaseControllerClass *base,GameObject *purchaser,unsigned int
cost,unsigned int preset,unsigned int purchaseret,const char *data) {
    if ((purchaseret == 0) || (purchaseret == (unsigned int)-1 && Commands->Get_Money(purchaser)
>= cost)) {
        std::string TransString = Translate_Preset_By_ID(preset);
        if (TransString == "No String") {
            FDSMessage(StrFormat("Purchase: %ls -
%ls",Get_Wide_Player_Name(purchaser),Translate_Preset_ID_To_PT_String_Name(base->tea
m,preset)),"_PURCHASE");
        }
        else {
            FDSMessage(StrFormat("Purchase: %ls -
%s",Get_Wide_Player_Name(purchaser),TransString.c_str()),"_PURCHASE");
        }
    }
}
}
}

```

```

void Level_Loaded() {
    strncpy(Data->CurrMap,The_Game()->MapName,29);
    Settings->Load();
}

```

```

Attach_Script_All_Buildings_Team(2,"MDB_SSGM_Building","",true);
Attach_Script_All_Turrets_Team(2,"MDB_SSGM_Base_Defense","",true);

```

```

if (Settings->EnableNewCrates) {
    Crate_Level_Loaded();
}
if (Settings->Gamelog) {
    Gamelog_Level_Loaded();
}
if (Settings->GameMode == 2) {
    CTF_Level_Loaded();
}
}

```

```

if (Settings->LogPlayerPurchase) {
    Data->PlayerPurchaseHookID = AddCharacterPurchaseMonHook(SSGM_Purchase_Hook,0);
}
}

```

```

if (Settings->LogPowerupPurchase) {
    Data->PowerupPurchaseHookID = AddPowerupPurchaseMonHook(SSGM_Purchase_Hook,0);
}
if (Settings->LogVehiclePurchase) {
    Data->VehiclePurchaseHookID = AddVehiclePurchaseMonHook(SSGM_Purchase_Hook,0);
}

if (!Data->Plugins.empty()) {
    std::vector<PluginInfo*>::const_iterator it;
    for (it = Data->Plugins.begin();it != Data->Plugins.end(); ++it) {
        if ((*it)->Type == Plugin) {
            if ((*it)->LevelLoadedHookHandle) {
                (*it)->LevelLoadedHookHandle();
            }
        }
    }
}

void GameOver() {
    char ObjectsType[10],ObjectsType2[10],ObjectsFile[20];
    Settings->INI->Get_String("General","ObjectsFile","ddb",ObjectsType2,10);
    Settings->INI->Get_String(The_Game()->MapName,"ObjectsFile",ObjectsType2,ObjectsType,10)
    ;
    sprintf(ObjectsFile,"objects.%s",ObjectsType);
    int FID = Commands->Text_File_Open(ObjectsFile);
    if (!FID) {
        FDSMessage(StrFormat("Failed to load %s file for map %s. Defaulting to
objects.ddb.",ObjectsFile,The_Game()->MapName),"_ERROR");
        sprintf(ObjectsType,"ddb");
        FID = Commands->Text_File_Open("objects.ddb");
        if (!FID) {
            FDSMessage("Critical Error! Objects.ddb file not found. Exiting FDS.", "_ERROR");
#ifdef WIN32
            MessageBox(HWND_DESKTOP,"Objects.ddb not found!","Critical SSGM
Error",MB_OK|MB_ICONEXCLAMATION);
#endif
            exit(1);
        }
        else {
            Commands->Text_File_Close(FID);
        }
    }
    else {
        Commands->Text_File_Close(FID);
    }
    Change_Objects_File(ObjectsType);
}

```

```

char WinType[50];
if (The_Game()->WinType == 3) sprintf(WinType,"Building Destruction");
else if (The_Game()->WinType == 2) sprintf(WinType,"High score when time limit expired");
else if (The_Game()->WinType == 4) sprintf(WinType,"Pedestal Beacon");
else if (The_Game()->WinType == 0) sprintf(WinType,"Server Shutdown");
else sprintf(WinType,"Unknown");
FDSMessage(StrFormat("Current game on map %s has ended. Game was won by %ls by
%s.",Data->CurrMap,Get_Wide_Team_Name(The_Game()->WinnerID),WinType),"_GENERAL");
if (Settings->Gamelog) {
    Gamelog_GameOver(WinType);
}
if (Settings->GameMode == 2) {
    CTF_GameOver();
}

Data->CrateExists = false;

if (Data->PlayerPurchaseHookID) {
    RemoveCharacterPurchaseMonHook(Data->PlayerPurchaseHookID);
    Data->PlayerPurchaseHookID = 0;
}
if (Data->PowerupPurchaseHookID) {
    RemovePowerupPurchaseMonHook(Data->PowerupPurchaseHookID);
    Data->PowerupPurchaseHookID = 0;
}
if (Data->VehiclePurchaseHookID) {
    RemoveVehiclePurchaseMonHook(Data->VehiclePurchaseHookID);
    Data->VehiclePurchaseHookID = 0;
}

if (!Data->Plugins.empty()) {
    std::vector<PluginInfo*>::const_iterator it;
    for (it = Data->Plugins.begin();it != Data->Plugins.end(); ++it) {
        if ((*it)->Type == Plugin) {
            if ((*it)->GameOverHookHandle) {
                (*it)->GameOverHookHandle();
            }
        }
    }
}

void Console_Output_Hook(const char* Output) {
    if (!Data->Plugins.empty()) {
        std::vector<PluginInfo*>::const_iterator it;
        for (it = Data->Plugins.begin();it != Data->Plugins.end(); ++it) {
            if ((*it)->Type == Plugin) {
                if ((*it)->ConsoleOutputHookHandle) {

```

```

    (*it)->ConsoleOutputHookHandle(Output);
}
}
}
}
}

//Type = 1(_CHEAT) or 2(_CHEATBAN)
DLLEXPORT void DragonGuard_Hook(int Type,GameObject *obj,const std::string &Header,const
std::string &Msg) {
    FDSMessage(Msg,Header);
}

//*****
//***** ORPHANS *****
//*****

#ifdef WIN32
HSZ ServiceName;
HSZ TopicName;
HSZ ItemName_Command;
DWORD Inst = 0;
HDDEDATA EXPENTRY FDSDDECallback(UINT wType, UINT fmt, HCONV hConv, HSZ hsz1,
HSZ hsz2, HDDEDATA hData, DWORD dwData1, DWORD dwData2) {
    switch (wType) {
    case XTYP_CONNECT: {
        if (!DdeCmpStringHandles(hsz1,TopicName) && !DdeCmpStringHandles(hsz2,ServiceName)) {
            return (HDDEDATA)TRUE;
        }
        else {
            return (HDDEDATA)FALSE;
        }
    }
    case XTYP_POKE: {
        unsigned char *DataRead = DdeAccessData(hData,NULL);
        bool SendToConsole = true;
        if (!Data->Plugins.empty()) {
            std::vector<PluginInfo*>::const_iterator it;
            for (it = Data->Plugins.begin();it != Data->Plugins.end(); ++it) {
                if ((*it)->Type == Plugin) {
                    if ((*it)->DDEHookHandle) {
                        SendToConsole = (*it)->DDEHookHandle((const char*)DataRead);
                    }
                }
            }
        }
        if (SendToConsole) {

```



```

OkLoad = true;

sprintf(Data->CurrMap,"");
Settings->InitINI();
Settings->LoadSString(Settings->FDSLogRoot,"FDSLogRoot","ssgm",true,false,false);

Console_Output("Server Side Game Manager v%s with Scripts.dll v%s
loaded\n",SSGMVersion,ScriptsVersion);
Console_Output("Created by Black-Cell.net\n");

char *Map = newstr(FirstMap().c_str());
char *Prefix = strtok(Map,"_");
if (!strcmp(Prefix,"C&C") || (Map[0] == 'M' && strlen(Map) == 7)) {
    Data->Mod = 0;
}
else if (!strcmp(Prefix,"TS")) {
    Data->Mod = 2;
}
else if (!strcmp(Prefix,"RA")) {
    Data->Mod = 3;
}
else {
    Data->Mod = 1;
}
delete[] Map;
if (Data->Mod) {
    FDSMessage(StrFormat("This server appears to be running %s. Some features will be
disabled.",Data->Get_Mod_Name().c_str()),"_GENERAL");
}

#ifdef WIN32
std::string Name;
Settings->LoadSString(Name,"DDEName","0",true,false,false);
if (Name != "0") {
    DdeInitialize(&Inst,FDSDDecallback,APPCLASS_STANDARD,0);
    ServiceName = DdeCreateStringHandle(Inst,Name.c_str(),0);
    TopicName = DdeCreateStringHandle(Inst,"FDSCommand",0);
    ItemName_Command = DdeCreateStringHandle(Inst,"Command",0);
    DdeNameService(Inst,ServiceName,0,DNS_REGISTER);
    Console_Output("%s DDE channel initialized\n",Name.c_str());
}
#endif

if (ChatCommandList::List) {
    Data->Copy_Chat_Commands(ChatCommandList::List);
}

for (int a = 1; ; ++a) {

```



```

char value[500];
Settings->INI->Get_String("Plugins",StrFormat("%02d",a).c_str(),"NULL",value,500);
if (!strcmp(value,"NULL")) {
    break;
}
#ifdef WIN32
HMODULE PluginHandle = LoadLibrary(StrFormat("%s.dll",value).c_str());
int LastError = GetLastError();
int LastError2 = 0;
if (!PluginHandle) { //Just in case the user included the file extension in the plugin list.
    PluginHandle = LoadLibrary(value);
    LastError2 = GetLastError();
}
#else
void *PluginHandle = 0;
#ifdef RH8
    PluginHandle = dlopen(StrFormat("./%s-RH8.so",value).c_str(),RTLD_NOW);
#else
    PluginHandle = dlopen(StrFormat("./%s-RH73.so",value).c_str(),RTLD_NOW);
#endif
if (!PluginHandle) { //Just in case the user included the file extension in the plugin list.
    PluginHandle = dlopen(StrFormat("./%s",value).c_str(),RTLD_NOW);
}
#endif
if (!PluginHandle) {
    Console_Output("Failed to load plugin %s: Error loading file.\n",value);
#ifdef WIN32
    std::ofstream ErrFile(StrFormat("Plugin_%s_Error.txt",value).c_str());
    if (ErrFile) {
        char *ErrorMessage = new char[2048];
        char *ErrorMessage2 = new char[2048];
        FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM,NULL,LastError,0,ErrorMessage,2048,NULL);
        FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM,NULL,LastError2,0,ErrorMessage2,2048,NULL);
        ErrFile << value << std::endl << ErrorMessage << std::endl << ErrorMessage2 << std::endl;
        delete[] ErrorMessage;
        delete[] ErrorMessage2;
    }
    ErrFile.close();
#endif
}
else {
    PluginInfo *Info = new PluginInfo;
    if (Info->Load(PluginHandle,value)) {
        Data->Plugins.push_back(Info);
    }
    else {

```

```

    delete Info;
}
}
}
std::ofstream VerINI("./ssgm_version.ini");
VerINI << "[Version]" << std::endl;
VerINI << "SSGM=" << SSGMVersion << std::endl;
VerINI << "Scripts=" << ScriptsVersion << std::endl;
VerINI << "BHS=" << BHS_VERSION << std::endl;
VerINI.close();
#ifdef WIN32
}
#endif
}

```

//This is called in Set_Script_Commands. Any code that requires scripts or bhs.dll functions is put here.

```

void SSGM_Secondary_Load() {
if (OkLoad) {
    AddChatHook(Chat);
    AddHostHook(HostChat);
    AddPlayerJoinHook(Player_Join_Hook);
    AddPlayerLeaveHook(Player_Leave_Hook);
    AddLoadLevelHook(Level_Loaded);
    AddGameOverHook(GameOver);
    AddConsoleOutputHook(Console_Output_Hook);

    Data->ObjectHookStruct = new ObjectCreateHookStruct;
    Data->ObjectHookStruct->hook = ObjectHookCall;
    Data->ObjectHookStruct->data = 0;
    Data->ObjectHookID = AddObjectCreateHook(Data->ObjectHookStruct);

    char ObjectsType[10],ObjectsFile[20];
    LoadSettingStringFirstMap(ObjectsType,10,"ObjectsFile","ddb",true,true);
    sprintf(ObjectsFile,"objects.%s",ObjectsType);
    int FID = Commands->Text_File_Open(ObjectsFile);
    if (!FID) {
        FDSMessage(StrFormat("Failed to load %s file for map %s. Defaulting to
objects.ddb.",ObjectsFile,The_Game()->MapName),"_ERROR");
        sprintf(ObjectsType,"ddb");
        FID = Commands->Text_File_Open("objects.ddb");
        if (!FID) {
            FDSMessage("Critical Error! Objects.ddb file not found. Exiting FDS.,"_ERROR");
#ifdef WIN32
            MessageBox(HWND_DESKTOP,"Objects.ddb not found!","Critical SSGM
Error",MB_OK|MB_ICONEXCLAMATION);
#endif
            exit(1);

```

```

}
else {
    Commands->Text_File_Close(FID);
}
}
else {
    Commands->Text_File_Close(FID);
}
Change_Objects_File(ObjectsType);

std::ofstream("gamelog2.txt").write("",0);
}
}

//Called when scripts.dll/.so is unloaded.
//Frees memory and unloads any plugins.
void SSGM_Unload() {
    if (!Data->Plugins.empty()) {
        std::vector<PluginInfo*>::const_iterator it;
        for (it = Data->Plugins.begin(); it != Data->Plugins.end(); ++it) {
            if (*it) {
                (*it)->Unload();
                delete *it;
            }
        }
    }
    Data->Plugins.clear();

    if (ChatCommandList::List) {
        std::vector<ChatCommandInfo*>::const_iterator it;
        for (it = ChatCommandList::List->begin(); it != ChatCommandList::List->end(); ++it) {
            if (*it) {
                delete (*it)->Ptr;
                delete *it;
            }
        }
        ChatCommandList::List->clear();
        delete ChatCommandList::List;
    }
    Data->Commands.clear();

    if (OkLoad) {
        RemoveObjectCreateHook(Data->ObjectHookID);
        delete Data->ObjectHookStruct;
    }
    delete Data;
    delete Settings;
}

```

```

//*****
//***** CHAT COMMANDS *****
//*****

class BindChatCommand : public ChatCommandClass {
void Triggered(int ID,const TokenClass &Text,int ChatType) {
    GameObject *Own = Get_GameObj(ID);
    GameObject *Veh = Get_Vehicle(Own);
    GameObject *MyVeh = Find_My_Veh(Own);
    if (!Veh) {
        Console_Input(StrFormat("ppage %d You must be the driver of a vehicle to use this
command.",ID).c_str());
    }
    else if (Get_Vehicle_Occupant(Veh,0) != Own) {
        Console_Input(StrFormat("ppage %d You cannot use this command as you are not the driver of
this vehicle.",ID).c_str());
    }
    else if (Get_Veh_Owner(Veh)) {
        Console_Input(StrFormat("ppage %d The vehicle you are trying to bind is already bound to
%ls.",ID,Get_Wide_Player_Name(Get_Veh_Owner(Veh))).c_str());
    }
    else {
        if (MyVeh) {
            Commands->Send_Custom_Event(Own,MyVeh,1111,0,0);
        }
        Commands->Attach_Script(Veh,"MDB_SSGM_Vehicle_Owner",StrFormat("%d,%d",Commands
->Get_ID(Own),Get_Object_Type(Own)).c_str());
        Console_Input(StrFormat("ppage %d Your vehicle has been bound to you. Any vehicle you
previously had bound has been unbound.",ID).c_str());
    }
}
};

ChatCommandRegistrant<BindChatCommand>
BindChatCommandReg("!bind",CHATTYPE_TEAM,0,GAMEMODE_AOW);

class UnbindChatCommand : public ChatCommandClass {
void Triggered(int ID,const TokenClass &Text,int ChatType) {
    GameObject *Own = Get_GameObj(ID);
    GameObject *MyVeh = Find_My_Veh(Own);
    if (!MyVeh) {
        Console_Input(StrFormat("ppage %d You must have a vehicle bound to you to use this
command. Type !bind in teamchat to bind the vehicle you are currently in.",ID).c_str());
    }
    else {
        Commands->Send_Custom_Event(Own,MyVeh,1111,0,0);
        Console_Input(StrFormat("ppage %d Your vehicle has been unbound.",ID).c_str());
    }
}
};

```

```

}
}
};
ChatCommandRegistrant<UnbindChatCommand>
UnbindChatCommandReg("!unbind",CHATTYPE_TEAM,0,GAMEMODE_AOW);

class LockChatCommand : public ChatCommandClass {
void Triggered(int ID,const TokenClass &Text,int ChatType) {
    GameObject *Own = Get_GameObj(ID);
    GameObject *MyVeh = Find_My_Veh(Own);
    if (!MyVeh) {
        Console_Input(StrFormat("ppage %d You must have a vehicle bound to you to use this
command. Type !bind in teamchat to bind the vehicle you are currently in.",ID).c_str());
    }
    else {
        Commands->Send_Custom_Event(Own,MyVeh,1112,0,0);
        Console_Input(StrFormat("ppage %d Your vehicle has been locked.",ID).c_str());
    }
}
};
ChatCommandRegistrant<LockChatCommand>
LockChatCommandReg("!lock",CHATTYPE_TEAM,0,GAMEMODE_AOW);

class UnlockChatCommand : public ChatCommandClass {
void Triggered(int ID,const TokenClass &Text,int ChatType) {
    GameObject *Own = Get_GameObj(ID);
    GameObject *MyVeh = Find_My_Veh(Own);
    if (!MyVeh) {
        Console_Input(StrFormat("ppage %d You must have a vehicle bound to you to use this
command. Type !bind in teamchat to bind the vehicle you are currently in.",ID).c_str());
    }
    else {
        Commands->Send_Custom_Event(Own,MyVeh,1113,0,0);
        Console_Input(StrFormat("ppage %d Your vehicle has been unlocked.",ID).c_str());
    }
}
};
ChatCommandRegistrant<UnlockChatCommand>
UnlockChatCommandReg("!unlock",CHATTYPE_TEAM,0,GAMEMODE_AOW);

class BLChatCommand : public ChatCommandClass {
void Triggered(int ID,const TokenClass &Text,int ChatType) {
    GameObject *Own = Get_GameObj(ID);
    GameObject *Veh = Get_Vehicle(Own);
    GameObject *MyVeh = Find_My_Veh(Own);
    if (!Veh) {
        Console_Input(StrFormat("ppage %d You must be the driver of a vehicle to use this
command.",ID).c_str());
    }
}
};

```

```

}
else if (Get_Vehicle_Occupant(Veh,0) != Own) {
    Console_Input(StrFormat("ppage %d You cannot use this command as you are not the driver of
this vehicle.",ID).c_str());
}
else if (Get_Veh_Owner(Veh)) {
    Console_Input(StrFormat("ppage %d The vehicle you are trying to bind is already bound to
%ls.",ID,Get_Wide_Player_Name(Get_Veh_Owner(Veh))).c_str());
}
else {
    if (MyVeh) {
        Commands->Send_Custom_Event(Own,MyVeh,1111,0,0);
    }
    Commands->Attach_Script(Veh,"MDB_SSGM_Vehicle_Owner",StrFormat("%d,%d",Commands
->Get_ID(Own),Get_Object_Type(Own)).c_str());
    Commands->Send_Custom_Event(Own,Veh,1112,0,0);
    Console_Input(StrFormat("ppage %d Your vehicle has been bound to you and locked. Any
vehicle you previously had bound has been unbound.",ID).c_str());
}
};
ChatCommandRegistrant<BLChatCommand>
BLChatCommandReg("!bl",CHATTYPE_TEAM,0,GAMEMODE_AOW);

```

```

class VKickChatCommand : public ChatCommandClass {
void Triggered(int ID,const TokenClass &Text,int ChatType) {
    GameObject *MyVeh = Find_My_Veh(Get_GameObj(ID));
    if (!MyVeh) {
        Console_Input(StrFormat("ppage %d You must have a vehicle bound to you to use this
command. Type !bind in teamchat to bind the vehicle you are currently in.",ID).c_str());
    }
    else {
        Force_Occupant_Exit(MyVeh,0);
        Console_Input(StrFormat("ppage %d The player in seat #1 of your vehicle has been
removed.",ID).c_str());
    }
}
};
ChatCommandRegistrant<VKickChatCommand>
VKickChatCommandReg("!vkick",CHATTYPE_TEAM,0,GAMEMODE_AOW);

```

```

class C4ChatCommand : public ChatCommandClass {
void Triggered(int ID,const TokenClass &Text,int ChatType) {
    Do_C4_Command(Get_GameObj(ID));
}
};
ChatCommandRegistrant<C4ChatCommand>
C4ChatCommandReg("!c4",CHATTYPE_TEAM,0,GAMEMODE_ALL);

```

its down under the chat hook.
